Architecture

HP 9000 V-Class Server

Second Edition

A3725-96022
HP 9000 V-Class Server
Customer Order Number: A3725-90004
March, 1998

Printed in: USA

Revision History

Edition: First

Document Number: A3725-96004
Remarks: Initial release October, 1997.

Edition: Second

Document Number: A3725-96022
Remarks: Released March, 1998.

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Contents

Figures

Tables

Preface

The document describes architecture of the Hewlett-Packard 9000 V-Class Enterprise Server based on the PA-8200, the latest in a line of high performance Precision Architecture - Reduced Instruction Set Computer (PA-RISC) processors from Hewlett-Packard Company.

New in Second Edition

The Second edition incorporates corrections to the First edition. The section entitled “Read Hint” has been removed from Chapter 2. A new section entitled “Accelerated Cache Coherence” has been added to Chapter 3.

System platforms

The HP-UX operating system is used on:

- K-Class servers
- D-Class servers
- V-Class servers

Notational conventions

This section describes notational conventions used in this book.

`bold monospace`	In command examples, **`bold monospace`** identifies input that must be typed exactly as shown.
`monospace`	In paragraph text, `monospace` identifies command names, system calls, and data structures and types. In command examples, `monospace` identifies command output, including error messages.
italic	In paragraph text, *italic* identifies titles of documents and provides emphasis on key words. In command syntax diagrams, *italic* identifies variables that you must provide. The following command example uses brackets to indicate that the variable *output_file* is optional: `command` *input_file* [*output_file*]
Brackets ([ ])	In command examples, square brackets designate optional entries.
Curly brackets ({}), Pipe (|)	In command syntax diagrams, text surrounded by curly brackets indicates a choice. The choices available are shown inside the curly brackets and separated by the pipe sign (|). The following command example indicates that you can enter either `a` or `b`: `command {a | b}`

Horizontal ellipses (...)	In command examples, horizontal ellipses show repetition of the preceding items.
Vertical ellipses	Vertical ellipses show that lines of code have been left out of an example.
Keycap	**Keycap** indicates the keyboard keys you must press to execute the command example.

NOTE A note highlights important supplemental information.

CAUTION A caution highlights procedures or information necessary to avoid damage to equipment, damage to software, loss of data, or invalid test results.

WARNING **Warnings highlight procedures or information necessary to avoid injury to personnel.**

1 Introduction

The V-Class server provides multipurpose, scalable computing resources though shared memory and I/O designed to provide high throughput and time to solution.

V-Class server uses the PA-8200, the latest in a line of high performance Precision Architecture - Reduced Instruction Set Computer (PA-RISC) processors from Hewlett-Packard Company.

The PA-8200 processor

The V-Class server uses the Hewlett-Packard PA-8200 processor, based on the concept of Reduced Instruction Set Computers (RISC). The PA-8200 was designed according to Hewlett-Packard's PA-RISC Architecture version 2.0 specifications.

NOTE

The PA-RISC architecture is presented in the *PA-RISC 2.0 Architecture* reference manual. Please refer to that document for detailed information about the features of the PA-8200. This document does not attempt to duplicate information in that manual. Instead, it presents only V-Class server-specific information.

The processors of the system are supported by several Application-Specific Integrated Circuit (ASIC) hardware controllers, an enhanced memory system, and a high-bandwidth I/O subsystem. Special hardware and software allow these processors to perform both as conventional single processors or together in parallel to solve more complex problems.

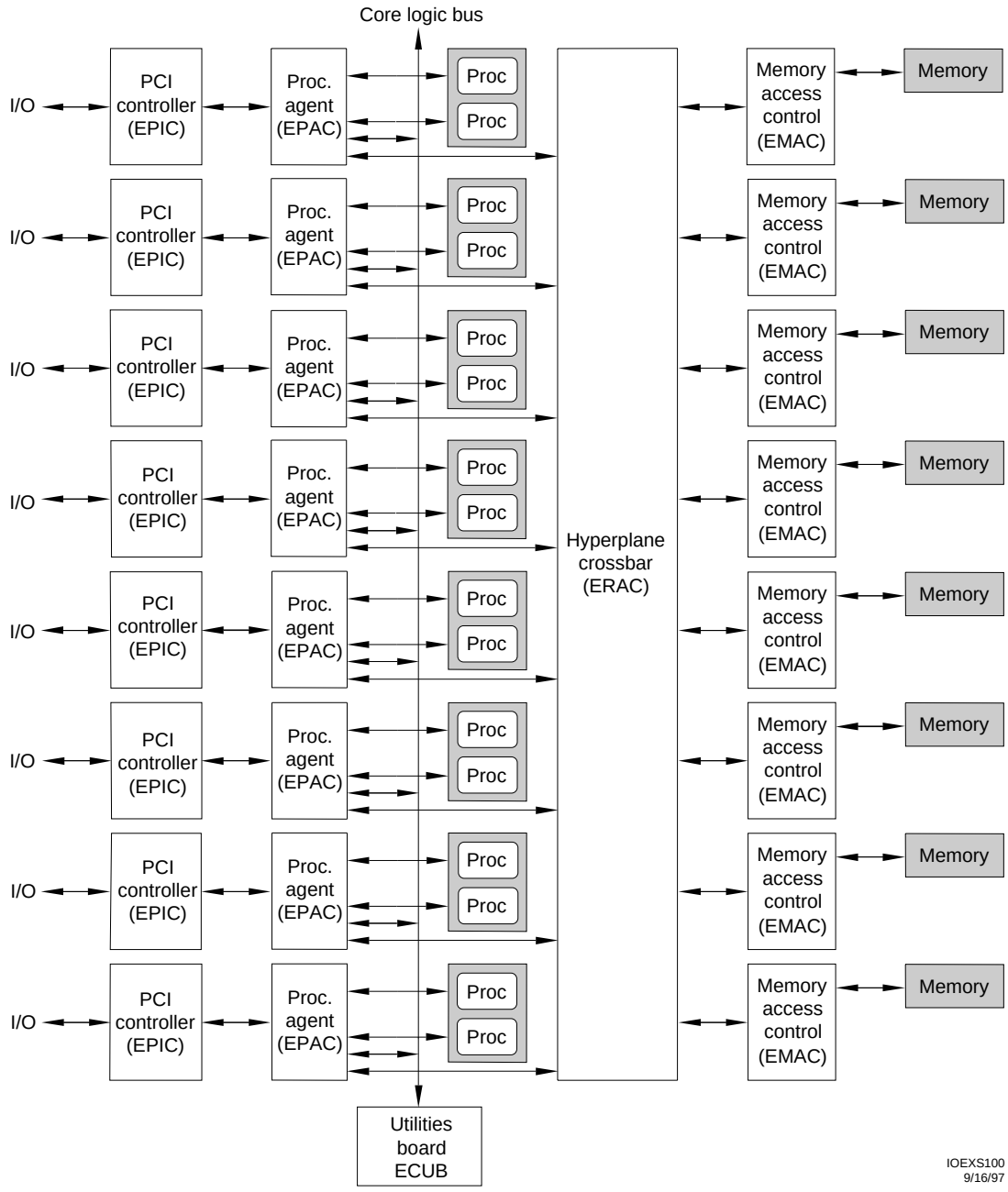
The node

The V-Class server can contain four to 16 processors. The processors and the associated hardware comprise what is commonly called node. The terms node and system are used interchangeably in this book. The node uses a symmetric multiprocessor (SMP) design that can exploit fine-grain parallelism.

A conceptual block diagram of the system is shown in Figure 1. Centrally located in the diagram is the HP Hyperplane crossbar that is comprised of four Exemplar Routing Attachment controllers (ERAC). The Hyperplane crossbar allows all of the processors to access all available memory. Processors are installed on Exemplar Processor Agent controllers (EPACs). An EPAC allows the processor and the I/O subsystem (the Exemplar PCI-bus Interface controller—EPIC) access to the Hyperplane crossbar. Also connected to the Hyperplane crossbar are the Exemplar Memory Access controllers (EMAC). Up to two processors are located on each EPAC. Memory is controlled by the EMAC. Input and output devices connect to the system through EPIC which is connected to the processor agents.

The Exemplar Core Utilities board (ECUB—commonly called the Utilities board) in the node contains a section of hardware called the core logic. It provides interrupts to all of the processors in the system through the core logic bus which connects to each processor agent. The ECUB attaches to the Exemplar system Routing board (ENRB) centrally located in the node.

Figure 1 Functional block diagram of a V-Class system



Control and status registers (CSRs)

System hardware is manipulated by control and status registers located in the processors and controllers.

CSRs provide control, status, or both to the processors and other hardware in the system. Each CSR is memory mapped and is available to all processors in the system. Many of the registers are described in detail by functional groups, such as system configuration, messaging and data copy, I/O, and so on. These descriptions appear throughout this book.

Description of functional blocks

Each block in Figure 1 is described in the following sections.

Exemplar processor agent controller

The EPAC can connect to zero, one, or two PA-8200 processors. It can also connect to zero or one EPIC (the I/O controller). With no processors, the EPAC serves as an I/O-only interface. The EPAC has the following buses:

- Runway bus (0, 1)—Two each, 64-bit, bidirectional buses for processor 0 and processor 1, respectively. These buses have a raw bandwidth of 960 MBytes per second.
- Hyperplane crossbar port bus (0, 1)—Four 32-bit, unidirectional buses connected to two Hyperplane crossbar ERACs, two in each direction. These buses have a total raw bandwidth of 1.9 GBytes per second.
- I/O port—Two 16-bit or 32-bit, unidirectional interfaces to an I/O device, one for reading data and one for writing data. The width of the bus depends on the width of the I/O device connected. Each bus has a bandwidth of 120 MBytes per second or 240 MBytes per second, depending on the width of the interface.
- Core Logic Bus interface—A single bidirectional bus that supports boot and support services.

The EPAC sends and receives transactions from the ERACs using four unidirectional data paths. There are four ERACs in the Hyperplane crossbar. Each processor agent, however, communicates with only two of the four ERACs.

The EPAC includes special hardware called the *data mover* for rapid message and data movement between memory within a node. This dedicated hardware greatly improves file I/O and networking over software versions.

Exemplar routing attachment controller—Hyperplane crossbar

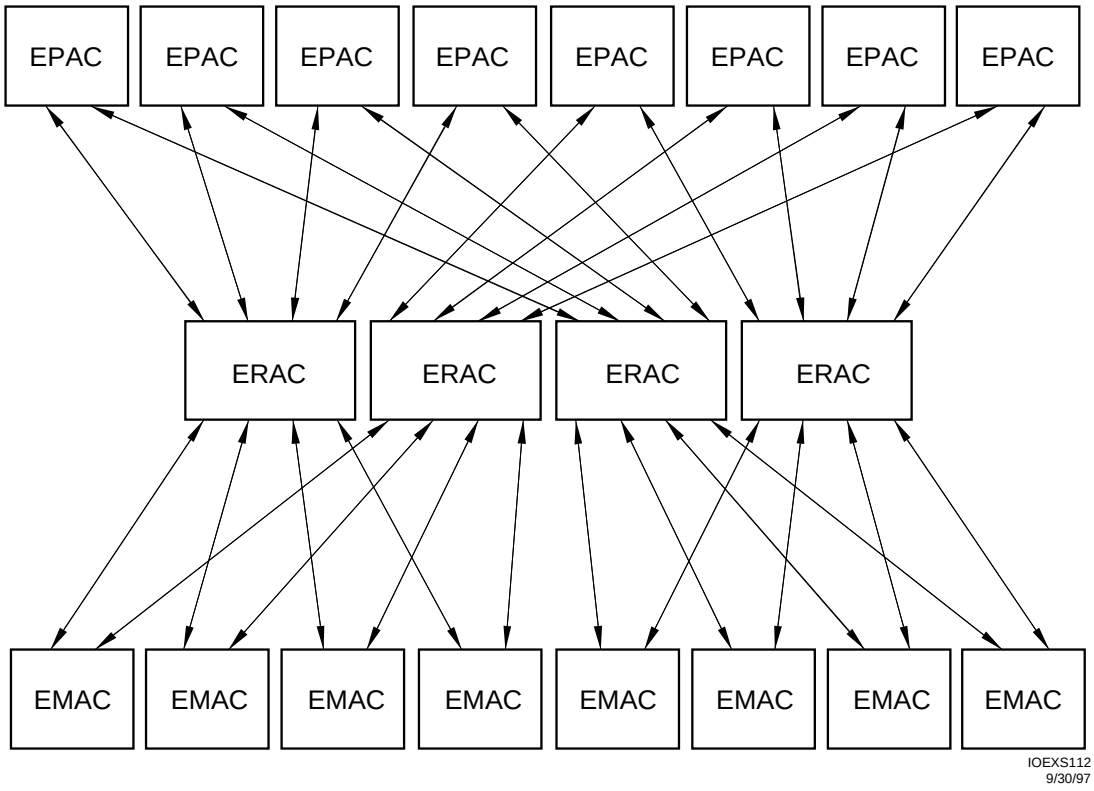
The Hyperplane crossbar is comprised of four ERACs that provide an interconnect for each processor and I/O device to memory.

Each of the four ERACs has the following buses:

- EPAC Port (A, B, C, D)—Eight 32-bit, unidirectional interfaces to four EPAC ports, four in each direction. Each port has simultaneous (input and output) bandwidth of 960 MBytes per second.
- EMAC Port (A, B, C, D)—Eight 32-bit, unidirectional interfaces to four EMAC ports, four in each direction. Each port has simultaneous (input and output) bandwidth of 960 MBytes per second.

Figure 2 shows how the ERACs connect to each EPAC and EMAC.

Figure 2 ERAC interconnection



Exemplar memory access controller

The EMAC controls all accesses to memory. Each EMAC controls four banks of memory, allowing up to 32 banks in an eight-EMAC system. Memory banks consist of Single Inline Memory Modules (SIMMs) of Synchronous Dynamic Random Access Memory (SDRAM).

The EMAC has the following buses:

- ERAC Port (A, B)—Four 32-bit, unidirectional interfaces, two in each direction. This interface supports a total simultaneous read-write bandwidth of 1.9 GBytes per second.
- Even Memory—A single 88-bit, bidirectional interface to the even memory banks associated with the EMAC.
- Odd Memory—A single 88-bit, bidirectional interface to the odd memory banks associated with the EMAC.

A processor accesses memory by sending a request, in the form of packets, to an ERAC. The request is then forwarded to one of the EMACs. The EMAC routes requests into even and odd pending queues. Some packets not destined for memory are routed from processor to processor through the EMAC. These packets are routed directly to the output ports.

The EMAC accesses one of four available memory banks, checking the Error Correction Code (ECC). The data accessed from memory is returned to the processor by sending a response back to the ERAC, which forwards the response to the EPAC.

ECUB and core logic bus

The ECUB, or Utilities board, connects to the core logic bus and contains two field-programmable gate arrays (FPGAs): the Exemplar Processor Utilities controller (EPUC) and the Exemplar Monitoring Utilities controller (EMUC). The EPUC allows processors access to the system core logic and booting firmware, and the EMUC processes the environmental state of the system and interrupts the processors when appropriate. V-Class servers use the core logic bus primarily to boot the system and to issue environmental interrupts.

The core logic bus is a low-bandwidth, multidrop bus that connects each processor to the control and interface logic (both RS232 and ethernet). A processor can write to control and status registers (CSRs) accessed using the core logic bus to initialize and configure the ERAC chips and Utilities board logic.

System configurations

Table 1 shows the available configurations.

Table 1 System configurations

Processors	Processor agents	Memory boards	Total memory (EMB)	I/O chassis
4	2	2	256	1
4	2	2	512	1
4	2	2	1024	1
4	2	4	2048	1
8	4	4	512	1 or 2
8	4	4	1024	1 or 2
8	4	4	2048	1 or 2
8	4	8	4096	1 or 2
12	6	8	1024	1, 2, or 3
12	6	8	2048	1, 2, or 3
12	6	8	3072	1, 2, or 3
12	6	8	4096	1, 2, or 3
16	8	8	1024	1, 2, 3, or 4
16	8	8	2048	1, 2, 3, or 4
16	8	8	3072	1, 2, 3, or 4
16	8	8	4096	1, 2, 3, or 4

Shared memory

V-Class servers use a shared-memory architecture to provide high-performance. This allows the developer, compilers, and applications to view the system as processors sharing a large physical memory and high-bandwidth I/O ports.

Message passing hardware provides ahigh performance for applications developed using a messaging scheme known as the Message Passing Interface (MPI). For more information concerning message passing, see the chapter entitled “Data mover,” for more information.

Compilers use shared memory to provide automatic, efficient parallelization, while viewing memory as a single contiguous virtual address space.

The Hyperplane crossbar provides high-bandwidth, low-latency nonblocking access from processors and I/O channels to the system memory. It prevents the performance drop-off associated with systems that employ a system-wide bus for processor and I/O memory traffic.

Sequential memory references (linearly ascending physical address) to shared memory are interleaved across up to eight memory boards on a 32-byte basis. See the chapter “Physical address space,” for more information.

With all processor references to memory, copies of the accessed data are encached into either the instruction or data caches of each processor. If the processor making the memory reference modifies the data and if another processor references that same data while a copy is still in the first processor cache, a condition exists whereby the data has become stale. The V-Class hardware continually works to ensure that the second processor does not use an outdated copy of the data from memory. The state that is achieved when both processors' caches always have the latest value for the data is called *cache coherence*.

To maintain updated coherent copies, V-Class servers operate under the following rules:

- Any number of read encachements of a cache line can be made at a single time. The cache line can be read-shared in multiple caches.
- To write (store) data into a cache line, the cache line must be “owned” exclusively by the processor. This implies that any other copies must be invalidated.
- Modified cache lines must be written back to memory from the cache before being overwritten.

2 Physical address space

This chapter describes the V-Class server physical address space, including coherent memory, core logic, and CSR address regions.

Physical addresses

The PA-8200 processor is an implementation of the 64-bit PA-RISC 2.0 architecture. The processor translates all 32- and 64-bit, virtual and absolute addresses to 64-bit physical addresses. External to the PA-8200 chip, however, only 40 bits of the 64-bit physical address are implemented.

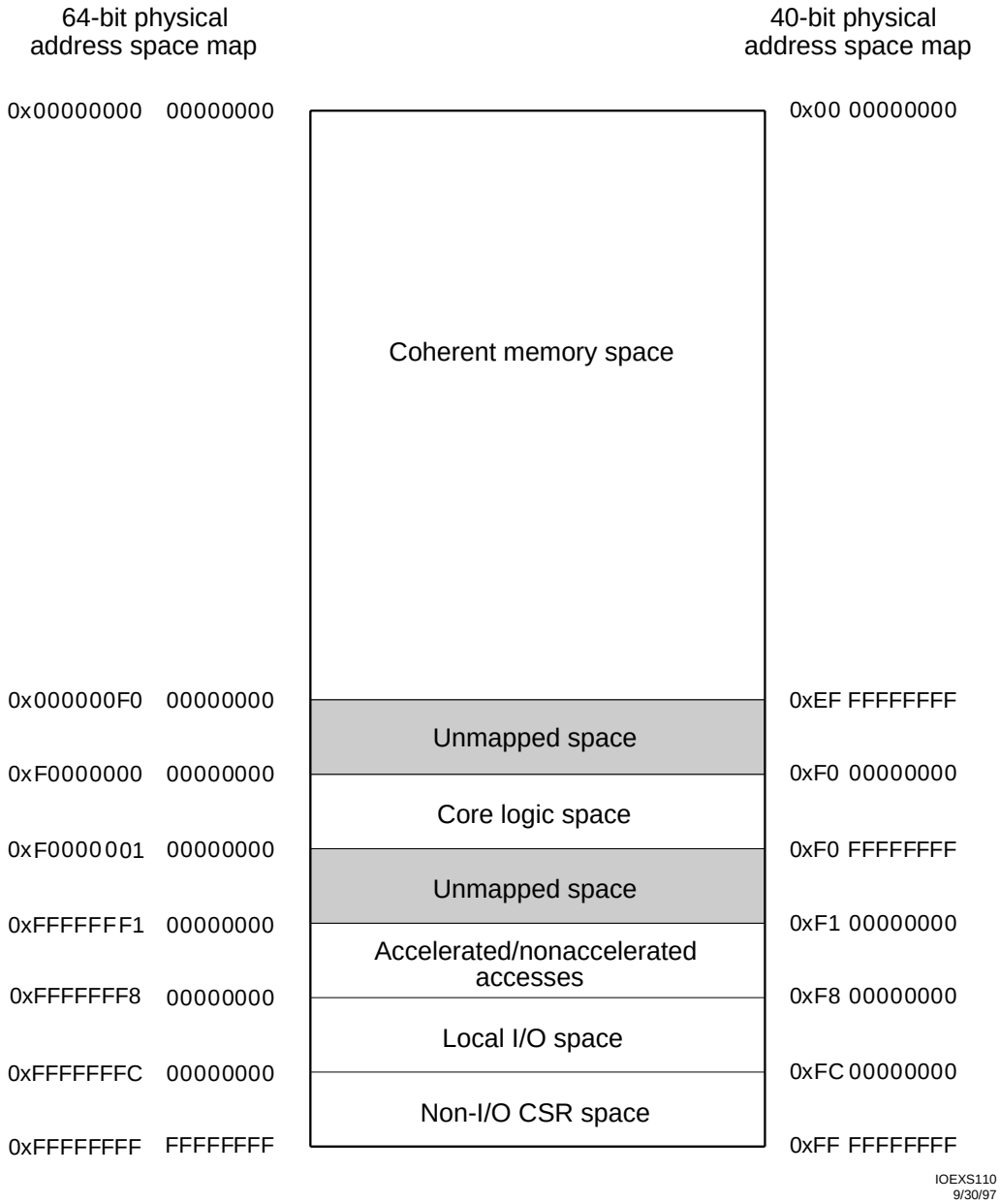
The I/O system uses controllers that have fewer than 40 address bits. The mapping of I/O addresses to the corresponding 40-bit physical address space occurs in the EPIC I/O subsystem.

V-Class server processors have four addressable physical address regions. These are:

- Coherent memory space—Memory used for programs and data and available to every processor. This is the bulk of the memory space.
- Core logic space—The space occupied by a group of hardware registers that comprises the system core logic function and is accessible to all processors within the system.
- Local I/O space—The space occupied by the PCI buses and prefetch and context RAM CSRs associated with input and output devices within the system.
- Non-I/O CSR space—The space occupied by the group of all other CSRs within the system.

Figure 3 shows how the address space is partitioned.

Figure 3 Physical address space partitioning

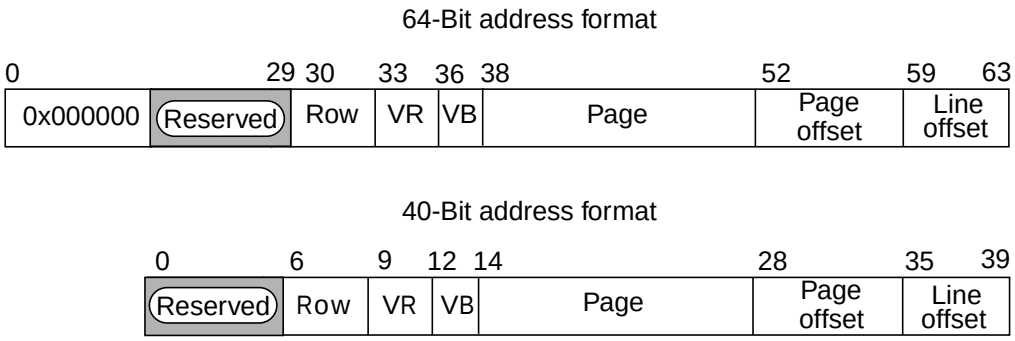


The left side shows the PA-8200 64-bit address map, and the right shows the 40-bit external address map used by the system. The two regions labeled unmapped space exist in the 64-bit physical address space but do not exist for the 40-bit physical address space.

Coherent memory space

As shown in Figure 3, coherent memory occupies the largest amount of physical address space. Figure 4 shows both the 64-bit and 40-bit physical address formats.

Figure 4 Coherent memory space address formats



The field definitions are as follows:

- *Row*—Selects one of eight rows of memory.
- *Virtual ring (VR)*—Selects one of eight memory boards.

NOTE

In V-Class systems, the term *ring* has the same meaning as memory board (MB). The term *ring* is used in this document to remain compatible with documentation of other similar servers.

- *Virtual bank (VB)*—Selects one of four memory banks.
- *Page*—Selects the page of memory.
- *Page offset*—Locates a line of memory within the selected page.
- *Line offset*—Locates the byte of memory within the selected line.

Coherent memory layout

Memory physically resides in memory blocks, with each block controlled by a single EMAC.

NOTE

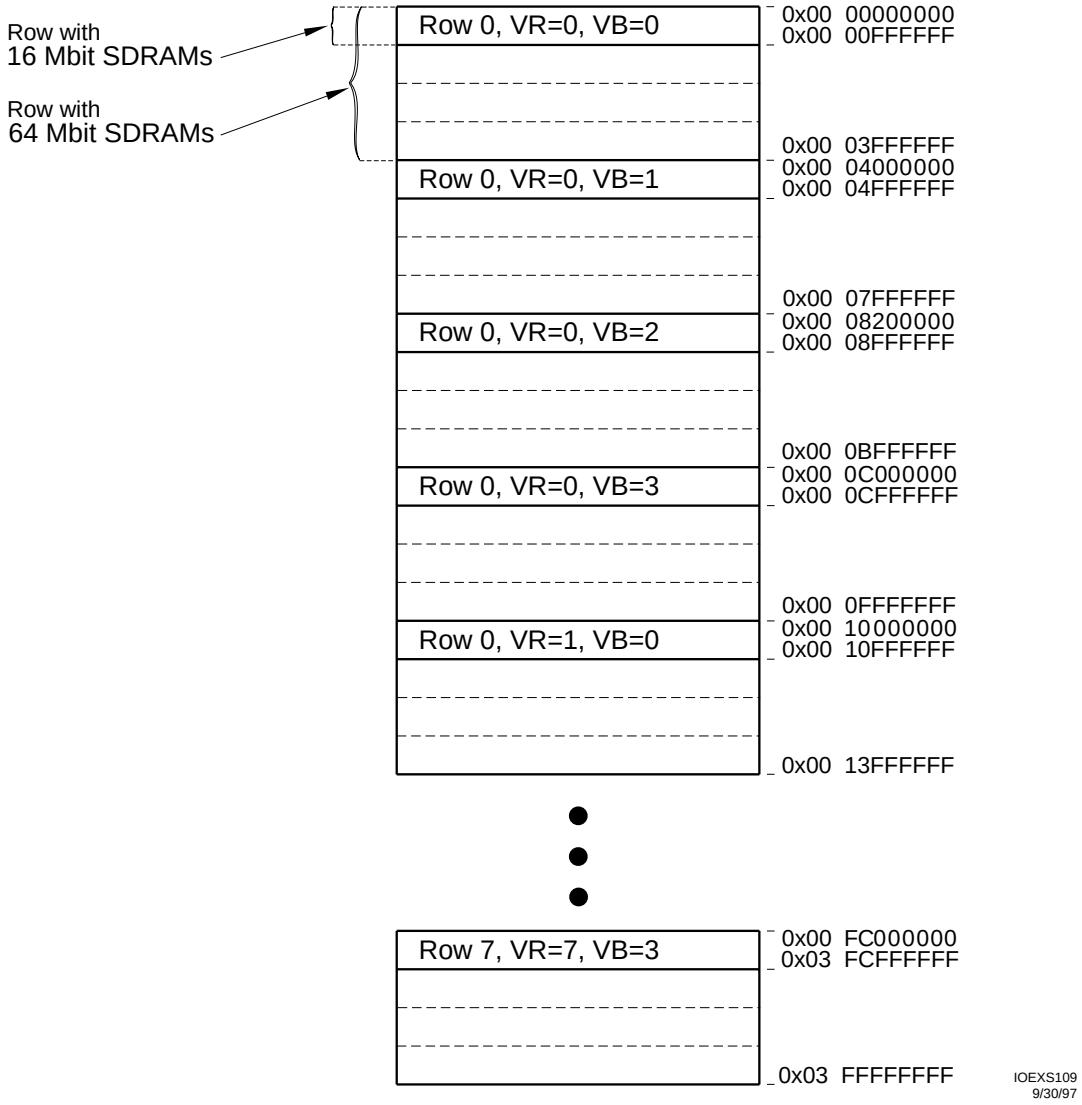
The term *memory block* is synonymous with memory board. There is a difference, however, in that a block is considered a logical entity and a board a physical entity. The EPAC maps logical memory blocks to physical memory boards.

Each block has four banks of memory. Coherent memory is further divided into memory lines 32bytes in size.

Memory blocks are implemented with memory DIMMs, up to 16 DIMMs per block (one block per EMAC). Each DIMM can have one or two rows of SDRAM chips, constructed with either 16-Mbit or 64-Mbit SDRAMs.

Figure 5 illustrates the layout for the coherent memory space.

Figure 5 Coherent memory space layout



Addressing a byte of memory

Figure 6 represents a fully populated system with 16 Gbytes of physical memory. It also shows how a byte of memory is addressed.

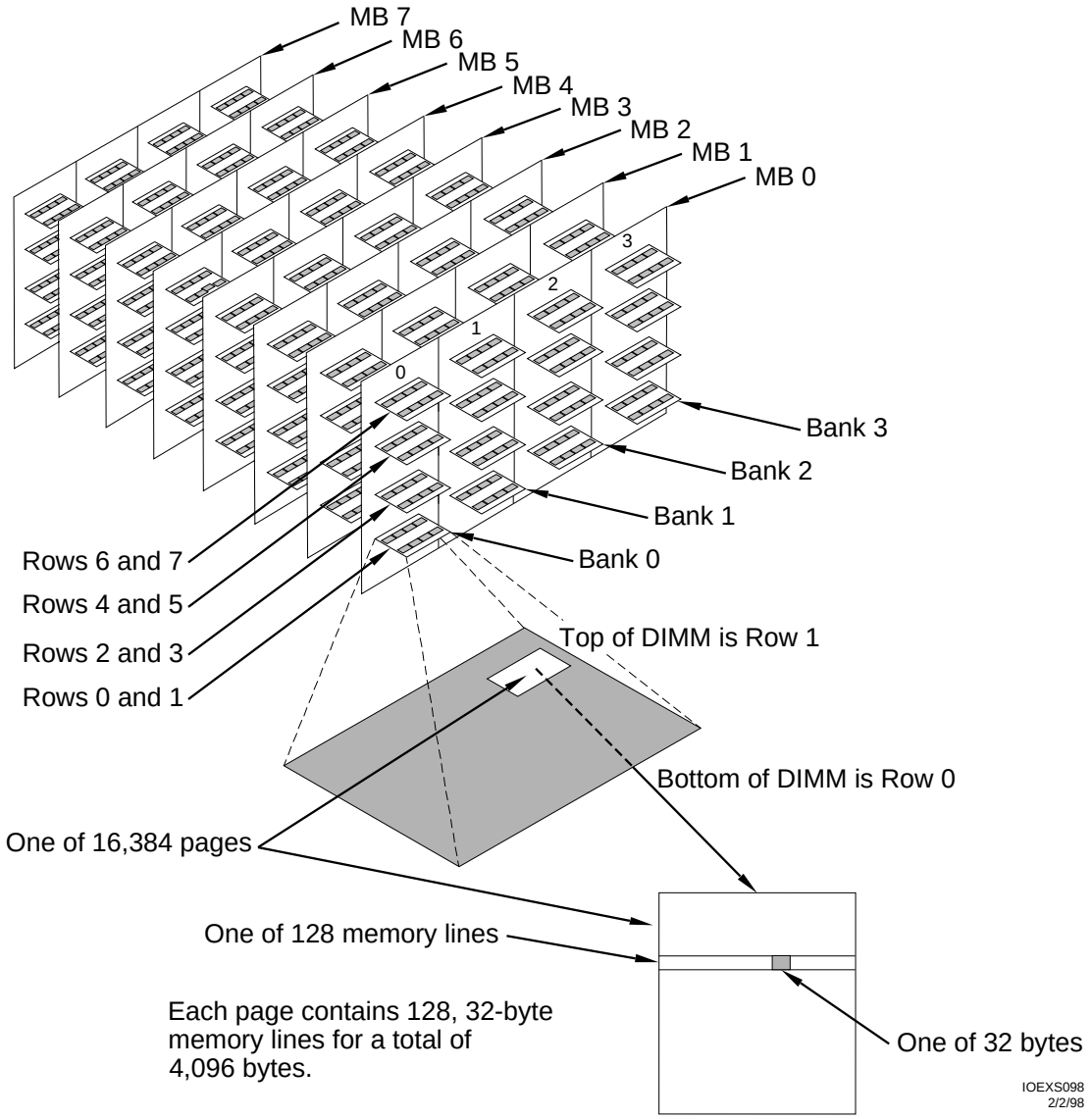
NOTE

Figure 6 represents only the concept of how memory is configured in the system. It does not depict the physical implementation.

The eight memory boards (MB) are at the top of the drawing. Each board has 16 DIMMS and each DIMM is loaded with memory chips on both sides. Each memory board has four banks comprised of four DIMMs in the vertical direction. Also, each board has eight rows along the horizontal direction. The top and bottom of each DIMM in the horizontal direction are part of two separate and adjacent rows. For example, Row 0 consists of the memory mounted on the bottom of each of the four DIMMs located on the bottom of the memory board in the horizontal direction. If the memory chips were 64-MBit SDRAM, each board would contain two GBytes of memory.

A pair of rows per bank (that is, rows 0 and 1) is sufficient to maintain maximum memory bandwidth. Additional rows add memory capacity, not additional bandwidth.

Figure 6 **Conceptual layout of physical memory of a fully populated system**



As shown in the physical address in Figure 4 and the conceptual memory layout in Figure 6, a byte of memory is accessed as follows:

1. The Row field selects one of eight rows of SDRAMs.
2. VR field selects the memory board.
3. VB field selects one of four banks on the appropriate memory board.
4. The Row, VR, and VB components of the physical address point to one side of a DIMM which contains 16,384 pages of 4,096 bytes each.
5. The Page field selects one of 16K pages on one side of the DIMM.
6. The Page offset field selects one of 128 memory lines in the page.
7. The Line offset field selects the appropriate byte in the line.

Each row contains 512 Mbytes of physical memory with 16-Mbit SDRAMs or 2 Gbyte with 64-Mbit SDRAMs. Within a row, 32 subpartitions exist, one for each memory board-bank combination (eight memory boards with four banks per board). Each subpartition is 64 Mbytes in size. If 16-Mbit SDRAMs are installed into a row of memory, then only the first 16 Mbytes of each subpartition of a row is accessible. Otherwise, with 64-Mbit SDRAMs, the entire 64 Mbyte subpartition is accessible.

Memory interleaving

Memory interleaving distributes consecutive lines of memory across as many banks as possible. It is supported across four, eight, 16, and 32 banks of memory as follows:

- A single memory block (four banks)
- An even-odd pair of memory blocks (eight-way interleave)
- Two pairs of memory blocks (16-way)
- Four pairs of memory blocks (32-way)

As noted earlier, the term memory block is synonymous with memory board. When generating the interleave, the EPAC maps logical memory blocks to physical memory boards. Therefore, describing memory interleave should be done in terms of memory blocks.

Normal memory interleave supports only *pairs* of memory blocks. Three pairs of memory blocks are interleaved with 16-way interleave.

Noninterleaved memory accesses are different from interleaved. They are divided into eight *rows*.

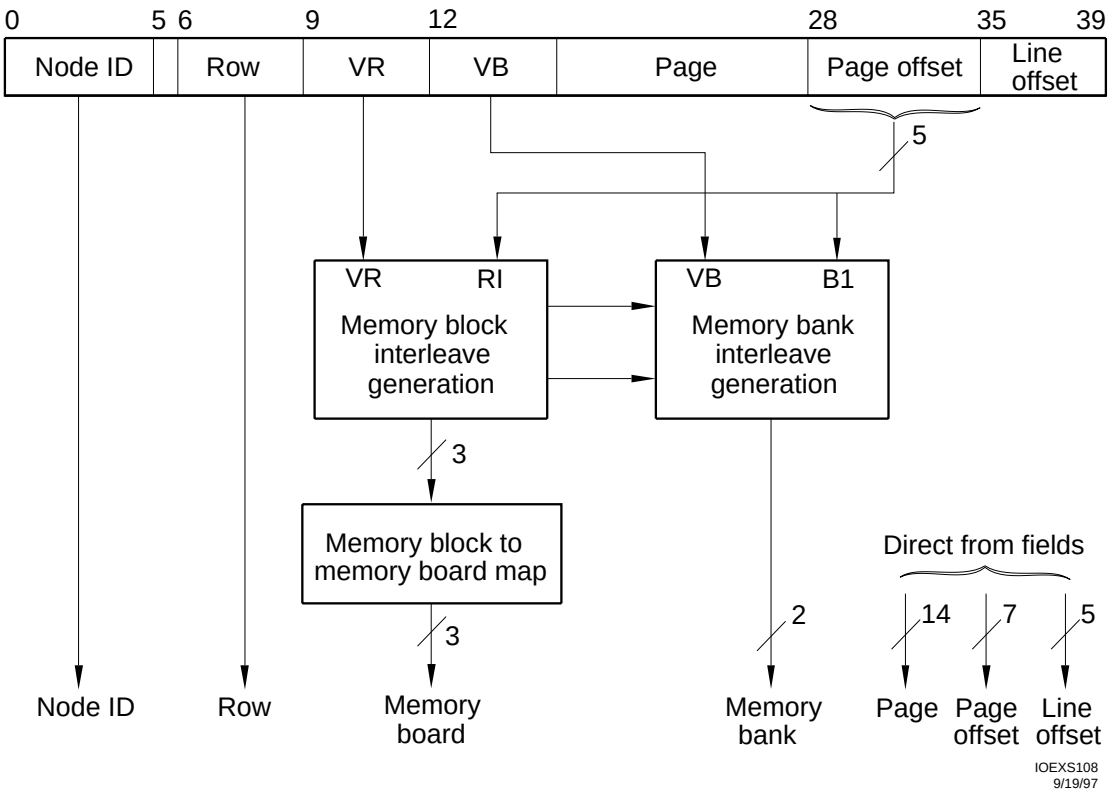
Memory interleave generation

Coherent interleave is performed on all memory references, except in the single memory block mode. To optimize memory bandwidth, memory blocks are installed on pairs of memory boards, even and odd. There are a maximum of four even-odd pairs of memory blocks in a system.

Figure 7 shows the mechanism for interleave. The 40-bit physical address provides the basis from which the memory blocks and memory bank are selected.

The Memory Board Configuration register supplies the map used to translate a memory block to the associated memory board where the memory line resides. See the section “EPAC Memory Board Configuration register” on page 39 for more information.

Figure 7 40-bit coherent memory address generation



The EMAC online field of the System Configuration register checks that a valid VR value is specified in the memory address. An invalid VR value results in an HPMC.

Ring (memory block) and bank index selection

The Page offset bits indicate the bank index and ring index. As noted earlier, the term ring means memory block—the two terms are synonymous in V-Class systems. If no memory interleaving is performed, the ring index is zero. Table 2 shows which offset bits are used. The numbers inside the parentheses indicate the appropriate address bits.

Table 2 **Bank/ring index selection**

Block pairs	Bank index (BI)	Ring index (RI)
No Interleave	Offset 33:34)	RI=0
One Pair	Offset (32:33)	Offset (34)
Two or Three Pairs	Offset (31:32)	Offset (33:34)
Four Pairs	Offset (30:31)	Offset (32:34)

Memory block interleave pattern

The memory block generated for the noninterleaved case is simply the VR field.

The memory blocks generated for interleave cases of one, two, and four board pairs are given in Table 3, Table 4, and Table 5, respectively. The tables show the memory board number with respect to the virtual ring and ring index.

Table 3 **Memory block interleave pattern for one pair**

VR (9:11)	RI=0	RI=1
0	0	1
1	1	0
2	2	3
3	3	2
4	4	5
5	5	4
6	6	7
7	7	6

Table 4 **Memory block interleave pattern for two pairs**

VR (9:11)	RI=0	RI=1	RI=2	RI=3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2
4	4	5	6	7
5	5	6	7	4
6	6	7	4	5
7	7	4	5	6

Table 5 **Memory block interleave pattern for four pairs**

VR (9:11)	RI=0	RI=1	RI=2	RI=3	RI=4	RI=5	RI=6	RI=7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

Memory bank interleave pattern

Memory bank interleaving occurs for all interleave spans. Memory configurations allow either two or four memory banks per EMAC. To support these two configuration options, two- and four-way memory bank interleaving are supported. Table 6 shows the memory bank interleave pattern for four banks.

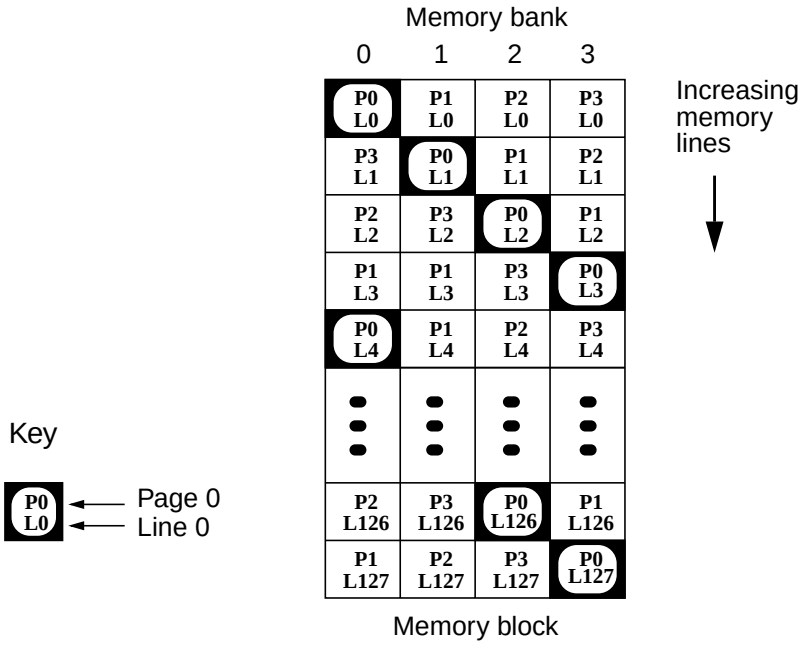
Table 6 **Memory bank interleave pattern for four banks**

VB (12:13)	BI=0	BI=1	BI=2	BI=3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Bank interleaved memory pattern

Single memory board interleave mode forces contiguous memory lines to reside in the same memory block. Within the memory block, memory lines interleave across the four memory banks. Figure 8 shows the interleave pattern for this mode. The pattern is the same independent of the number of memory blocks in the system. The pattern shown is for a 4096-byte page of memory (128 memory lines per page).

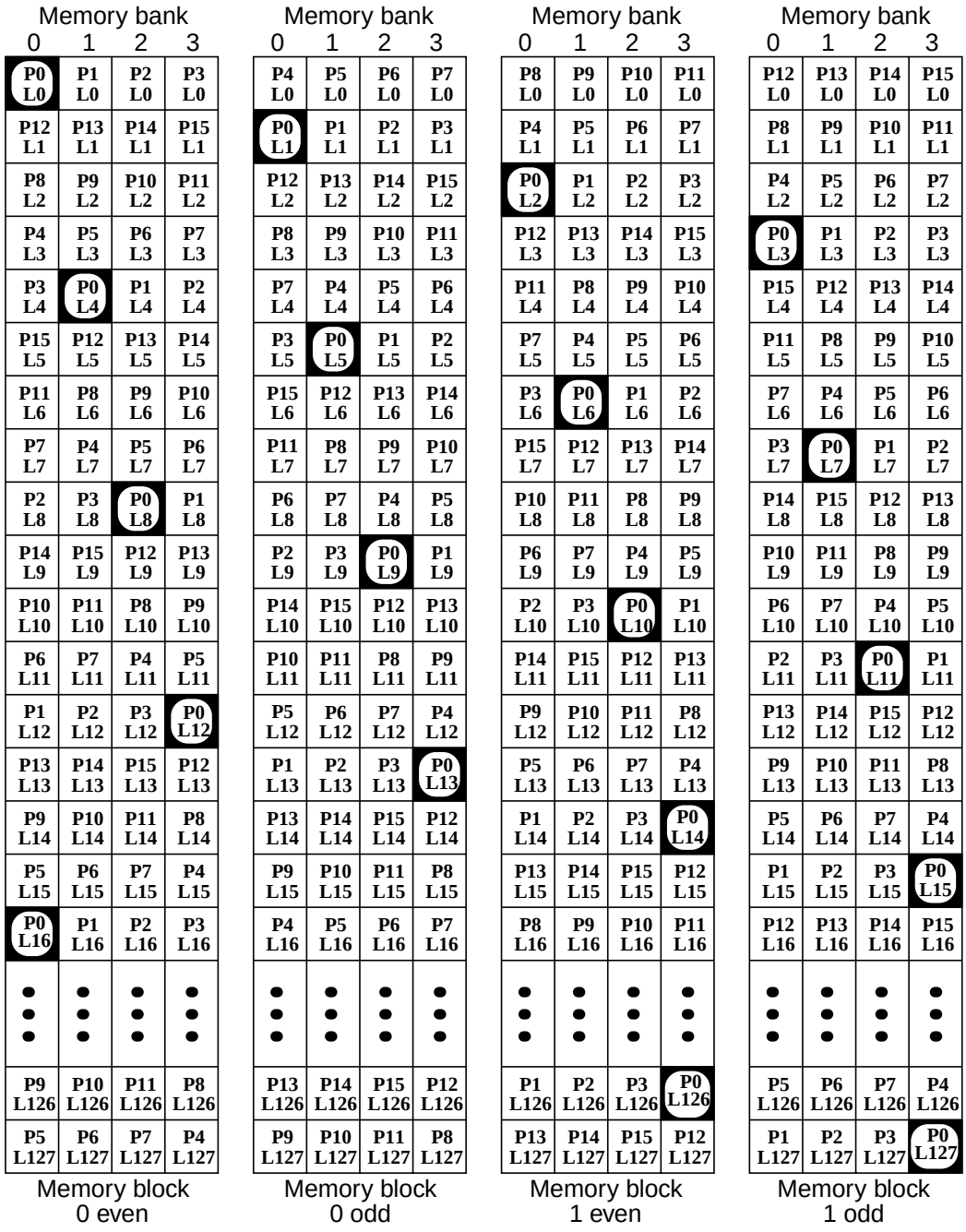
Figure 8 **Single memory block interleave pattern**



Block and bank interleave memory pattern

In multiple memory board interleave modes, memory lines are interleaved across the largest power-of-two memory banks. There are up to eight memory blocks, resulting in 32-way memory line interleaving. The minimum system configuration has two memory blocks, resulting in eight-way memory line interleaving. With both four and six memory blocks in the system, the interleave is 16-way. Figure 9 shows the interleave pattern for a system with four memory blocks. The pattern shown is for 4096-byte pages of memory (128 memory lines per page).

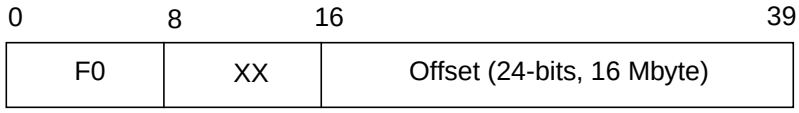
Figure 9 Memory line interleave pattern with four memory blocks



Core logic space

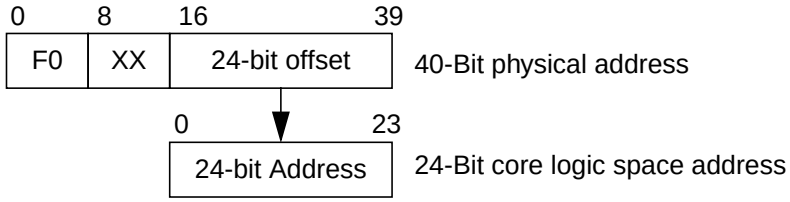
Core logic space is used to access core logic hardware (EEPROM, SRAM, and core logic CSRs) and is only accessible within the system. The processor has a fixed decode for this space. Bits 8 through 15 of the 40-bit physical address are ignored for address decoding.

Figure 10 **40-bit core logic space format**



The addresses contain a 24-bit offset used as the core logic bus address. The EPAC translates from the physical addresses to core logic bus addresses. It also splits 64-bit requests into two, 32-bit requests. The EPUC translates the core bus address to utility address. See Figure 11.

Figure 11 **Core logic address translation**



Core logic space is further partitioned for EEPROM, SRAM, and CSR Space. Table 7 shows the address ranges for each of these partitions.

Table 7 **Core logic space partitions**

Partition	Core logic space offset range
EEPROM	0x000000 - 0x7FFFFF
SRAM	0x800000 - 0xBFFFFF
CSR	0xC00000 - 0xFFFFFF

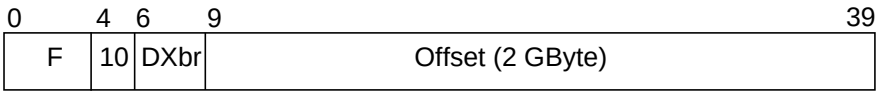
Processor-dependent code (PDC) space is accessed using the core logic bus attached to each processor. A PDC space access is not routed through the Hyperplane crossbar.

Local I/O space

A processor can directly access all I/O space in the system using the I/O controllers.

The EPAC of the source processor checks the value of the DXbr field against the appropriate bit of the Processor Agent online field in the System Configuration register of the source processor to verify that the destination processor agent is online. The EPAC of the destination processor checks to see if the EPIC online bit in its Chip Configuration register is set. If either of the online bits are not set, the request will fail with a high-priority machine check trap. See Figure 12.

Figure 12 40-bit local I/O space format



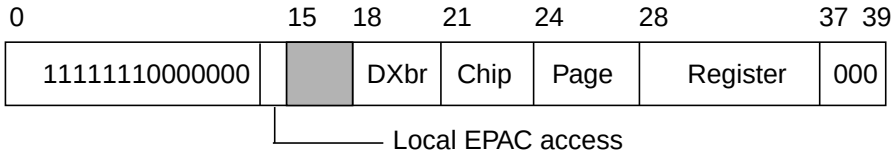
The bits of the local I/O space address are as follows:

- *DXbr* field (bits 6:9)—Specifies to which of the eight Hyperplane crossbar ports (connected to the EPAC chip) the request is to be routed.
- *Offset* field (bits 33:39)—Specifies the offset into I/O space. In this space, all PCI configuration, I/O CSRs and I/O memory space must be allocated. See the section “Host-to-PCI address translation” on page 106 for more information.

Non-I/O CSR space

Non-I/O CSRs reside within the PA-8200, EPAC, EMAC, and EPIC.

Figure 13 **Non-I/O CSR space format**



The bits and fields of the CSR space address are as follows:

- *Local EPAC access bit* (bit 14)—Indicates that the access is to the local EPAC space within the associated EPAC ASIC.
- *DXbr* field (bits 18:19)—Specifies which of the eight Hyperplane crossbar ports the request is to be routed.
- *Chip* field (bits 21:23)—Routes the packet to the appropriate chip at a Hyperplane crossbar port.
- *Page* field (bits 28:36)—Separates groups of CSRs into similar usage spaces.
- *Register* field (bits 28:36)—Specifies the register number.

CSR access

There are three packet routing methods used for accessing CSRs:

- Processor-local
- EPAC-local
- Node

The 40-bit physical address determines which access method will be used.

Processor-local access

Processor-local accesses reference CSRs that reside in the processor issuing the request. These accesses are sent out and brought back into the requesting processor on its Runway bus. The EPAC, which is also connected to the Runway bus, ignores the request. The processor online bits of the processor agent are not checked for processor-local accesses.

EPAC-local access

EPAC-local accesses are accesses to CSRs that reside in the EPAC physically connected to the processor that is issuing the request. These accesses are identified as EPAC-local and are not sent to the Hyperplane crossbar. Table 8 shows which fields must be specified for EPAC-local addressing.

This method accesses processor-specific CSRs that reside in an EPAC. All processor-specific EPAC CSRs are identified as having bit 2 of the Chip field set. When the EPAC detects a processor-specific page, it forces bit three of the page field according to the processor issuing the request.

NOTE The EPAC online field of the System Configuration CSR is not checked for EPAC-local accesses.

System accesses

System accesses are used to access CSRs throughout the system. If the access is to an EMAC, the DXbr field routes the request to the proper EMAC. The EMAC Online field of the source EPAC System Configuration register is checked to ensure the destination EMAC is online. A high-priority machine check trap will result if the destination EMAC is not online.

If the access is to an EPAC, EPIC, or processor, the request is first routed to the EMAC specified by the Intermediate EMAC field of the EPAC Configuration CSR. The EPAC Online field of the source EPAC System Configuration CSR is checked to ensure the destination EPAC is online. If the destination EMAC is not online, an HPMC results.

Table 8 Field specifications for system access

Field	Specification
Bits 0:5	0x3F
Local EPAC	0
SXbr	X
DXbr	Destination Hyperplane crossbar port
Chip	Destination chip

Access to nonexistent CSRs

It is possible to send a request to a CSR in a controller that is not online. Online bits are implemented for processors, EPACs, EMACs, and EPICs. Memory uses existence bits.

Accesses to nonexistent CSRs terminate in one of the following ways:

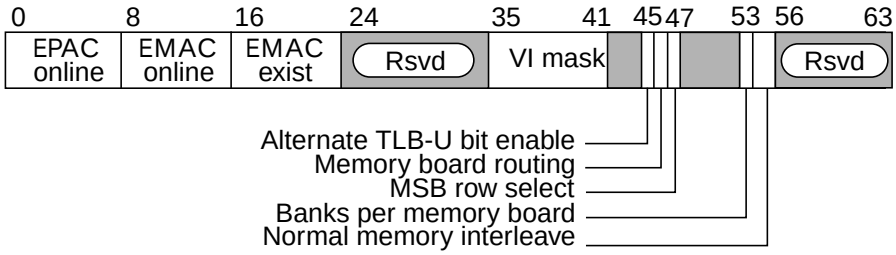
- Requests with a response to a CSR covered by an online bit result in an error response being returned to the processor. The processor issues a high-priority machine check interrupt.
- Requests without a response to a CSR covered by an online bit result in a time-out when the next synchronization operation is performed. The synchronization time-out results in a high-priority machine check interrupt.
- Requests with a response to a CSR not covered by an online bit result in a time-out. The request time-out results in a high-priority machine check interrupt.
- Requests without a response to a CSR not covered by an online bit result in a time-out when the next synchronization operation is performed. The synchronization time-out results in a high-priority machine check interrupt.

System Configuration register

The System Configuration register specifies system configuration parameters. The register is replicated on the EPAC and EMAC, but only fields used by each controller type are implemented for a particular controller. Therefore, not all fields exist on an EPAC or EMAC.

Figure 14 shows the generic format of the register. All fields are written to by a write access and read from by a read access. All fields are unaffected by reset unless specified.

Figure 14 **System Configuration register definition**



The bits and fields of the System Configuration register are defined as follows:

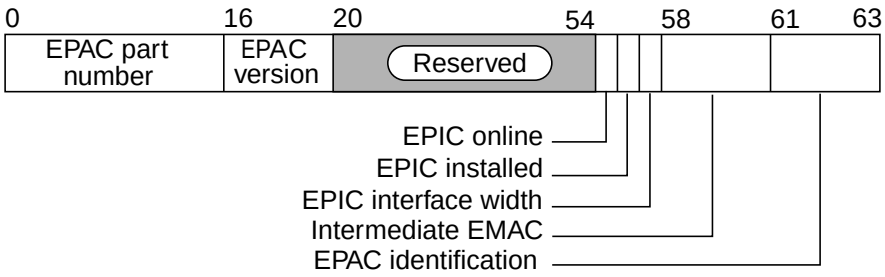
- *EPAC online* field (bits 0:7)—Specifies which EPACs are accessible. These bits are used to validate all I/O space and local CSR Space requests. The field is cleared by reset.
- *EMAC online* field (bits 8:15)—Specifies which EMAC ASICs are accessible. These bits are used to validate all Coherent Memory Space and CSR Space requests. The field is cleared by reset.
- *EMAC exist* field (bits 16:23)—Indicates which EMAC ASICs exist in the system. These bits are used by software to initialize the EMAC online field. The field is initialized by reset. A CSR write is ignored.
- *VI mask* field (bits 35:41)—Specifies the Virtual Index bits generated by the PA-8200 processor that are masked (forced to zero).
- *Alternate TLB-U bit enable* bit (bit 45)— Enables checking for coherent accesses to noncoherent semaphore memory.
- *Memory board routing* bit (bit 46)—Selects the coherent memory request to the memory board routing function. When the bit is zero, even coherent memory requests are routed to even memory boards and odd requests to odd boards. When the bit is one, even coherent memory requests are routed to odd memory boards and odd requests to even boards.
- *MSB row select* bit (bit 47)—Selects which bit of the 40-bit physical address is the most significant bit of the three-bit row selection information.
- *Banks per memory board* bit (bit 53)—Specifies whether two or four banks exist per memory board.
- *Normal memory interleave* field (bits 54:55)—Specifies the number of even/odd memory board pairs which normal interleaving should span.

EPAC Configuration register

Each EPAC has one Processor Agent Configuration register which specifies information about the EPAC. Each EPAC can be configured differently.

Figure 15 shows the format of the EPAC Configuration register. All fields of the register are read by a read access.

Figure 15 EPAC Configuration register definition



The bits and fields in the EPAC Configuration register are defined as follows:

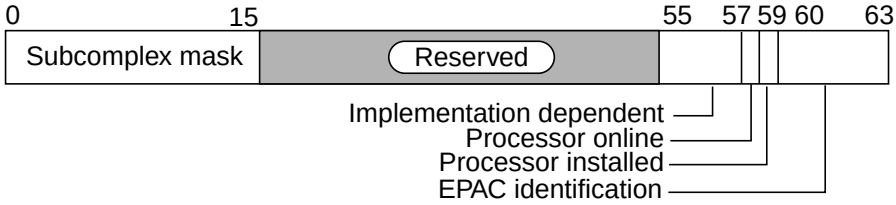
- *EPAC part number* field (bits 0:15)—Specifies the part number for the EPAC. A write is ignored and a read returns the hard-wired value.
- *EPAC version* field (bits 16:19)—Specifies the version for the EPAC. A write is ignored and a read returns the hard-wired value.
- *EPIC online* bit (bit 55)—Set by software to allow CSR accesses to the EPIC. The bit is cleared by reset.
- *EPIC installed* bit (bit 56)—Specifies whether an EPIC ASIC is connected to the EPAC. A value of one indicates an EPIC is installed. This bit is read only.
- *EPIC interface width* bit (bit 57)—Specifies whether a 32-bit or 16-bit interface exists between the EPIC and EPACs. A value of one indicates a 32-bit interface, a value of zero indicates 16-bit. This bit is read only.
- *Intermediate EMAC* field (bits 58:60)—Specifies the physical EMAC used by the EPAC when routing a packet to another EPAC. Any EMAC installed in the system can be specified and packet routing will function properly.

- *EPAC identification* field (bits 61:63)—Specifies the identification number for the physical EPAC. The value is obtained from pins on the EPAC. A write to this field is ignored and a read access will return the value of the pins.

EPAC Processor Configuration register

Each EPAC has a Processor Configuration register that contains specific information about the EPAC processor. Each processor attached to the EPAC can be configured differently. Figure 16 shows the format of the EPAC Processor Configuration register. All fields of the register are read by a read access.

Figure 16 EPAC Processor Configuration register definition



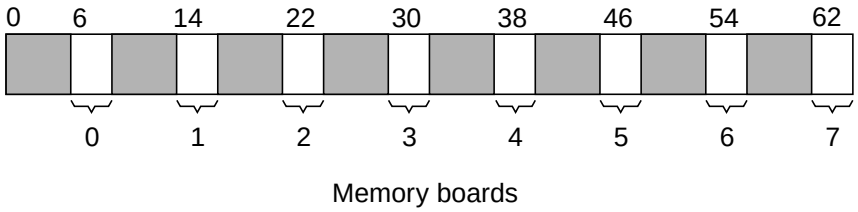
The bits and fields in the EPAC Processor Configuration register are defined as follows:

- *Subcomplex mask* field (bits 0:15)—Determines which processors should receive broadcasted transactions.
- *Implementation dependent* field (bits 55:57)—Used by low-level implementation dependent software. The value in this field should not be modified during normal operation.
- *Processor online* bit (bit 58)—Indicates that the processor is accessible. The value is initialized to the value of the processor installed bit.
- *Processor installed* bit (bit 59)—Indicates that the processor is installed. The value of this bit comes directly from a pin on the EPAC. Writes to this bit are ignored; a read access will return the value of the input pin.
- *Processor identification* field (bits 60:63)—Specifies the identification number for the physical processor. The value read is obtained by concatenating the three EPAC ID pins and bit 27 of the 40-bit address used to read the register. A write to this field is ignored and a read access will return the value of the pins/address bit.

EPAC Memory Board Configuration register

Each EPAC has a Memory Board Configuration register that specifies the memory block to memory board mapping. Figure 17 shows the format of the register. All fields are written by a write access and read by a read access. Reset has no effect. Writes to reserved bits are ignored and reads to reserved bits return the value zero.

Figure 17 EPAC Memory Board Configuration register definition



The three-bit memory block generated by the memory block interleave generation logic indexes into one of the eight memory board fields of the Memory Board Configuration register.

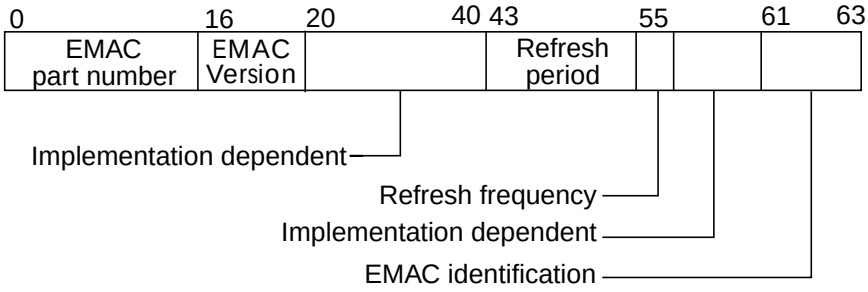
The memory board fields specify the most significant two bits of the physical memory board. The least significant bit of the memory block index is the least significant bit of the physical memory board. This forces even memory blocks to be mapped to even memory boards and odd memory blocks to odd memory boards.

EMAC Configuration register

Each EMAC has a Configuration register that contains specific information. Each EMAC can be configured differently.

Figure 18 shows the format of the register. All fields of the register are read by a read access.

Figure 18 EMAC Configuration register definition



The bits and fields in the EMAC Configuration register are defined as follows:

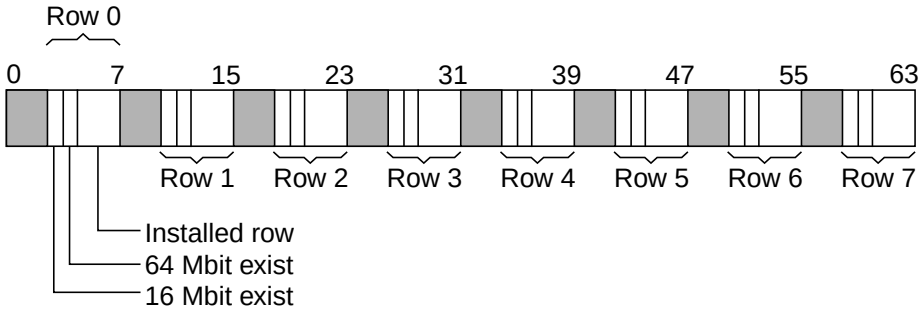
- *EMAC part number* field (bits 0:15)—Specifies the part number for the EMAC chip. A write is ignored and a read returns the hard wired value.
- *EMAC version* field (bits 16:19)—Specifies the version for the EMAC chip. A write is ignored and a read returns the hard wired value.
- *Implementation dependent* field (bits 20:42)—Used by low-level implementation dependent software. The value in this field should not be modified during normal operation.
- *Refresh period* field (bits 43:54)—Indicates how often refresh occurs. For SDRAMs that need to be refreshed every 15.6us, this value should be set to 0x3a8, which is 936 half clocks.
- *Refresh frequency* field (bits 55:56)—Controls the operation of refresh. The value of this field after reset is 3.
- *Implementation dependent* field (bits 57:60)—Used by low level implementation dependent software. The value of these bits should not be modified during normal operation.
- *EMAC identification* field (bits 61:63)—Specifies the identification number for the physical EMAC. The value is written by software.

EMAC Memory Row Configuration register

Associated with each EMAC are four banks of memory, each bank having up to eight rows of SDRAMs. A table maps the row specified in the physical address to a physically installed row of SDRAMs. The four banks of memory controlled by an EMAC have the same memory row mapping.

Each EMAC has a register that specifies the memory row mapping. The format of the register is shown in Figure 19. All fields are written by a write access and read by a read access. Reset has no effect. Writes to reserved bits are ignored and reads to reserved bits return the value zero.

Figure 19 Memory Row Configuration register definition



The 3-bit *row* field of a physical address indexes one of the eight sets of *row* fields of a Memory Row configuration register.

The bits and fields in the EMAC Memory Row Configuration Register are defined as follows:

- *16 Mbit exist* bits—Indicate that a DIMM with 16-Mbit SDRAMs exists for the row. The field checks memory existence access.
- *64 Mbit exist* bits—Indicate that a DIMM with 64-Mbit SDRAMs exists for the row. The field checks memory existence access.
- *Installed row* fields—Map the row specified by the physical address to the physical memory row.

3 Caches

Caches compensate for memory latency by storing frequently accessed memory lines, either data or instructions, locally on the processor board. Whenever the particular data or instruction is required again, the processor does not have to wait for it to return from the system memory.

Processor caches

Each processor has two caches: a data cache and an instruction cache, referred to as dcache and icache, respectively. These caches are two Mbytes in size each. The data cache may be modified; the instruction cache may not.

Whenever data is loaded into the data cache (this process is called *move-in*), the processor can modify it there. If another processor makes a data reference to the same item while the copy is still in the first processor's cache, hardware ensures that the original item and the copy are identical to maintain coherency.

Often more than one memory line is encached. For data references, all cache lines of the referenced page can be imported, and for instruction references, all lines in the referenced page and the following page (both virtual and physical) can be imported. A flush cache, purge cache, or purge TLB instruction stops any subsequent move-in operations to that page until another reference is made.

Each processor supports speculative execution of code, which is enabled with virtual address translation. Speculative execution allows cache move-in of any coherent memory line, provided that the virtual-to physical address translation of the memory line is in the processor translation lookaside buffer (TLB). Also, the virtual-to-physical translation bit in the processor status word must be set and the TLB entry must have the Uncacheable bit (U bit) cleared. See the section "PA-8200 TLB Entry U-bit" on page 88.

Caches are flushed by one of two methods:

- Specifying a memory line address
- Specifying the cache entry to flush

All cache operations are issued with a single processor instruction. These include:

- Flush Data Cache (FDC)
- Purge Data Cache (PDC)
- Flush Data Cache Entry (FDCE)
- Flush Instruction Cache (FIC)
- Flush Instruction Cache Entry (FICE)

The FDC and PDC instructions have identical functionality on the PA-8200. Both operations flush cache lines from the data caches of all processors in the system. FDC and PDC instructions write data from *dirty* cache lines back to memory.

The FDCE and FICE instructions flush an entry from the executing processor's cache only. If the cache line in the data cache is dirty, it is written back to local memory.

NOTE Always follow cache flush instructions by a `sync` instruction to ensure that all flushes are complete.

Cache coherence between processors

Cache coherence causes the system to behave as if it had a single data cache and a single instruction cache (logically) for all processors. Since there are many processors and, therefore, multiple data caches, each processor must cross-interrogate for current data and broadcast purges and flushes (except for FDCE and FICE).

All coherent data references are satisfied using cache coherence checks. These checks ensure that the data has remained coherent since it was moved in. Cache coherence checks are performed on write buffers in order to ensure proper ordering of storage accesses.

Accelerated cache coherence

V-Class servers employ the Multi-Level Runway bus (MLR) mode on PA-8200 processors for coherent memory transfers. This mode allows multiple runway buses in the same system to have a coherent view of memory.

In MLR mode during read requests, the processor does not take ownership of a memory line until the EPAC accesses memory and the EMAC sends a response back to the processor. When the processor receives the response, it checks to see if the line is in its outbound queue. If the processor does have the data in this queue, it issues a write back to the EPAC.

Accelerated cache coherence is employed, because some applications that do not take the MLR mode into account could use certain instruction sequences. These would degrade system performance by causing increased memory access latency and bandwidth.

Accelerated cache coherence hardware detects whether the processor could have any data in its outbound queue. If the outbound queue is empty, it tags the read request so that the EMAC determines the queue is empty and does not need to send a flush to the processor. Since the EMAC does not have to flush the queue, it can send the data to the processor immediately. Without accelerated cache coherence, the read transfer would require two memory accesses.

When several processors reference the same memory line, the EMAC for that line maintains a tag for each line of main memory. The tag keeps track of which processors are sharing the line and how they are accessing

it. Using this tag, the EMAC forwards transactions to a limited number of processors rather than to all of them. These tags are separate from the cache tags maintained by the processor.

Address aliasing

The *PA-RISC 2.0 Architecture* manual describes address aliasing. For a full discussion of PA-RISC address aliasing, refer to this document.

Two or more virtual addresses that map to the same physical address are called aliases. The PA-RISC architecture recognizes two types of aliases: equivalent and nonequivalent.

Equivalent aliasing

Equivalent aliases are virtual addresses for which offset bits 40 through 63 and space bits 36 through 47 are the same. This means that the offset portions of the addresses differ by a multiple of 16 Mbytes. A virtual address that is equal to the absolute address it maps is an *equivalent mapping*; this is a simple case of equivalent aliasing.

The PA-RISC architecture allows unrestricted equivalent aliasing. There may be any number of equivalent aliases, with any combination of mappings (read-only, writable, etc). The V-Class server architecture completely supports this type of aliasing.

Nonequivalent aliasing

Nonequivalent aliases do not satisfy the requirements for equivalent aliases. If nonequivalent aliases exist, the PA-RISC architecture requires that they must all be read-only. If a writable translation is required, any aliases that are not equivalent to the writable translation must be removed from the page table and flushed from the TLB before the translation is made writable.

NOTE The V-Class architecture does not support nonequivalent aliasing regardless of whether the aliases are read-only or not.

Nonequivalent aliasing on V-Class servers may cause a hardware virtual index error that results in an HPMC. Only processor coherent memory references cause virtual index errors; processor instruction fetches and references from I/O adapters can not cause these errors. The error only occurs when the following two conditions are met:

- The reference is a processor data reference.
- The referenced line may have been encached by another processor using an alias that is not equivalent to the current reference.

Nonequivalent aliasing can not be used on V-Class servers. Furthermore, to prevent the unintentional introduction of nonequivalent aliases, special cache flushing protocol must be observed when unmapping a physical page and remapping it at a different virtual address and when mixing absolute accesses and virtual accesses to the same page.

Before a page is remapped, it must be completely flushed from all data caches in the system.

This cache flushing must use FDC to avoid virtual index errors. Flushing the data cache with FDCE is not sufficient to avoid virtual index errors, because data may be left in other processors' caches. Even flushing the page with FDCE on every processor is insufficient, because the memory system maintains coherency tags that are unaffected by FDCE.

The virtual address of the FDC is irrelevant; only the absolute address to which it is mapped is important. This means that memory can be flushed to prevent virtual index errors without knowing how it was previously mapped.

Instruction aliasing

The V-Class server supports instruction aliasing. If none of the aliases allows data references (i.e., all aliases are executable only or allow no access at all), arbitrary aliasing is allowed, whether equivalent or not. The V-Class server handles data and instruction references completely separately, so there can be arbitrary instruction aliasing. Instruction cache coherence must be maintained in software.

NOTE Any aliases that allow data references must be equivalent aliases.

4 Data mover

The EPAC contains a hardware section called the data mover that routes messages and copies data between memory. This chapter describes how message transfer is accomplished.

Overview

In order for the V-Class server to process separate threads, it must send process context messages from one thread in a process to another within the same or different process. The messages and data shared between threads are transferred between processors by way of common memory (shared memory).

Message transfers

For message transfers, the message resides in the source memory location before the transfer and in destination memory location after the transfer. Source and destination addresses are specified with either a virtual address or a physical address.

Data copy

For a data copy operation, the data resides in the memory of both the source and destination location. The size of the data copy operation can vary, with the source and destination address being specified within a single page or with a list of pages.

Data mover features

The following list highlights the data mover hardware functions:

- Messaging and data copy can be to and from both interleaved and noninterleaved memory.
- Each processor has an independent interface for copy transfers.
- The initiating processor notifies the destination processor of impending messaging operations with queued completion status.
- The initiating processor receives a confirmation interrupt at the completion of the transfer.
- Copy operations can be from virtual-to-virtual, virtual-to-physical, physical-to-virtual, or physical-to-physical space.
- Messages can be performed from virtual or physical space.

- The source or destination space can be specified as a single page (where the page is up to four Mbytes in size), or with a Block Translation Table (BTT) with each entry mapping a four-Kbyte page.

Data mover implementation

The V-Class server uses CSRs in the EPACs and EMACs and memory addresses along with the data mover to transfer messages and data. These CSRs and addresses include:

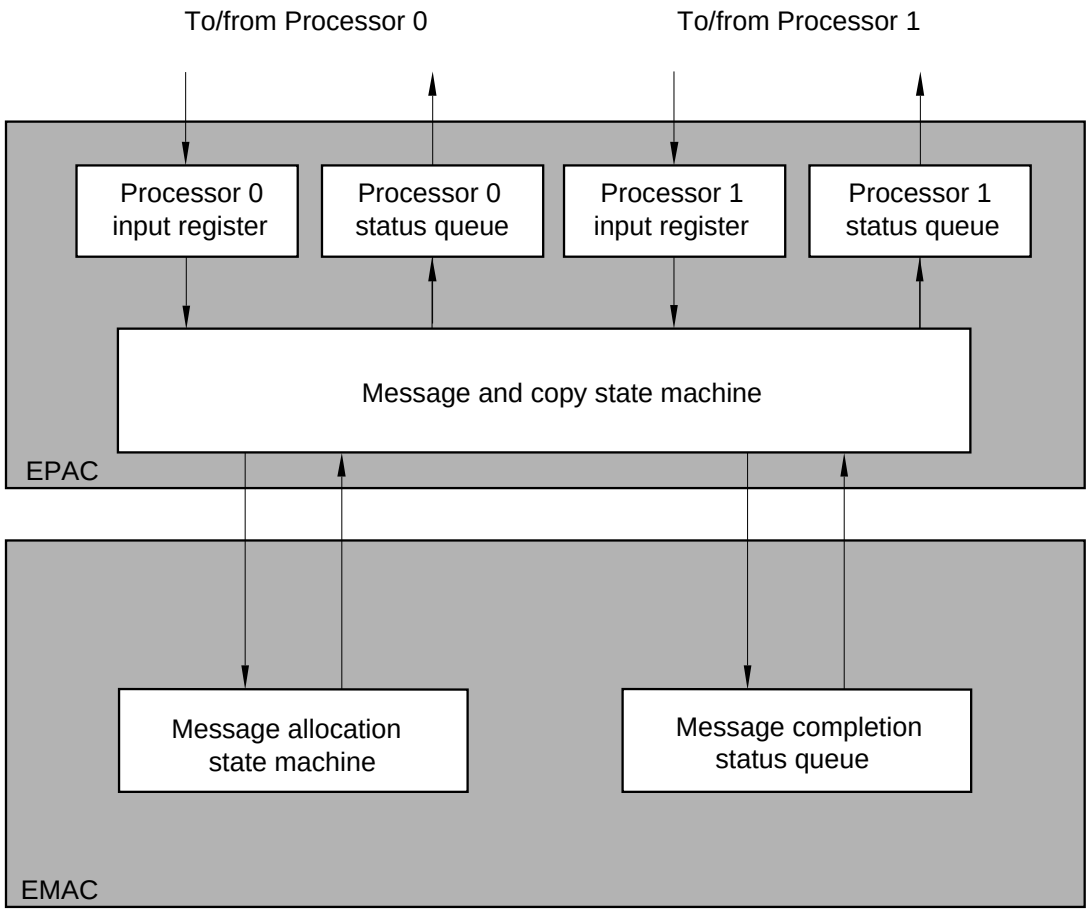
- EPAC Operation Context register
- EPAC Operation address register
- EPAC Input Command register
- EPAC Source and Destination Physical Page Frame registers
- EPAC Source and Destination Offset registers
- EPAC Operation Status Queue register
- EMAC Message Reception Area Configuration registers
- EMAC Message Reception Area Offset registers
- EMAC Message Completion Queue Configuration registers
- EMAC Message Completion Queue Offset registers
- EMAC Memory Allocation address
- EMAC Message Completion Enqueue address
- EMAC Message Completion Dequeue address

The implementation uses memory structures for message reception, message completion queues, and the BTT for I/O data copy transfers. Memory structures are preallocated regions of memory. The actual data resides in memory before and after transfer.

Functional overview

Figure 20 shows a functional diagram of the messaging and data copy transfers.

Figure 20 Messaging and data copy transfers implementation



Input registers

Each EPAC has two input registers for its two processors. To initiate a messaging or data copying operation, a processor first determines if the input registers are available. All transfer operations remain in the input register until the transfer begins. Once the input registers are available, a processor initiates an operation by programming the input registers. The processor can set up a second transfer in the input registers while the first transfer is in progress

Message and copy state machine

The message and copy state machine starts executing an input operation when a set of input registers has been set up and the message and copy state machine is idle. If both sets of input registers have operations ready to execute, the hardware arbitrates between the two sets of input registers to guarantee forward progress.

The state machine achieves the transfer operation by executing the following three phases:

- It determines the destination address for message operations. If the current operation is a copy operation, this phase of execution is skipped. The destination address is determined by sending a transaction to an EMAC. The EMAC performs a memory allocation operation and responds with a destination memory address.
- It copies data from the source memory to the destination memory. The copy operation executes until complete or until either a TLB purge or error occurs. For more information on TLBs, see the *PA-RISC 2.0 Architecture* manual.
- It sends a message completion transaction to the EMAC of the destination address. This phase is not performed if the operation is a data copy. The EMAC enqueues the completion status in a memory-based queue and informs the destination processor by way of an interrupt.

Operation status queues

The two processors connected to the EPAC each have an operation status queue. The state machine places the message and copy operation completion status in one of the appropriate status queues. Each status queue is three entries deep to provide status space for multiple overlapped operations.

Once status is enqueued, an interrupt is sent to the processor that initiated the operation.

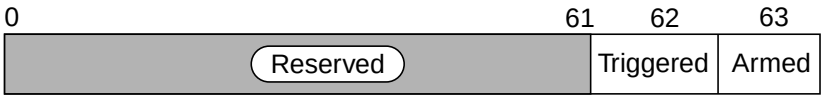
Messaging and data copy CSRs

CSRs in both the EPACs and EMACs control messaging and data copy. This section specifies the addresses used to access each CSR of the messaging and data copy hardware.

EPAC Operation Context registers

Each EPAC has two Operation Context registers, one for each processor. The operation context is applied to other CSRs in two ways. One is by arming a register, and the other is by indicating that the armed register was triggered, that is, it performed a specific function. Figure 21 shows the format of the EPAC CSR Operation Context register.

Figure 21 EPAC CSR Operation Context register definition



The bits of the CSR Operation Context register are defined as follows:

- *Triggered* bit (bit 62)—Indicates that a CSR operation executed when the Armed bit was set. The Triggered bit is cleared by software and is set by hardware.
- *Armed* bit (bit 63)—Set by software to arm the functionality of specific EPAC processor CSRs. The EPAC CSRs armed by this bit are:
  - Data Mover Input Command register
  - Fetch and Increment address
  - Fetch and Decrement address
  - Fetch and Clear address
  - Noncoherent Read address
  - Noncoherent Write address
  - Coherent Increment address

Each of these CSRs is discussed in this chapter. The Armed bit is set by software and is cleared by either hardware or software.

Table 9 shows the Armed and Triggered bit transitions that the hardware controls when software writes to one of the operation addresses.

Table 9 CSR Operation Context register transitions when the operation is issued

Present value		Next value	
Triggered	**Armed**	**Triggered**	**Armed**
0	0	0	0
0	1	1	0
1	0	0	0
1	1	1	1

Table 10 shows the Armed and Triggered bit transitions that hardware controls when a TLB invalidate transaction is detected.

Table 10 CSR Operation Context register transitions with TLB invalidate

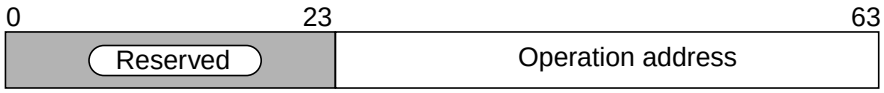
Present value		Next value	
Triggered	**Armed**	**Triggered**	**Armed**
0	0	0	0
0	1	0	0
1	0	1	0
1	1	1	1

EPAC Operation Address registers

Each EPAC has two Operation Address registers, one for each processor. The register stores the address used for CSR operations.

The format of the EPAC Operation Address register is shown in Figure 22. The field of the register is read by a read access.

Figure 22 EPAC Operation Address register definition



The Operation address field (bits 24:63) designates the address for CSR operations and also the Coherent Increment address.

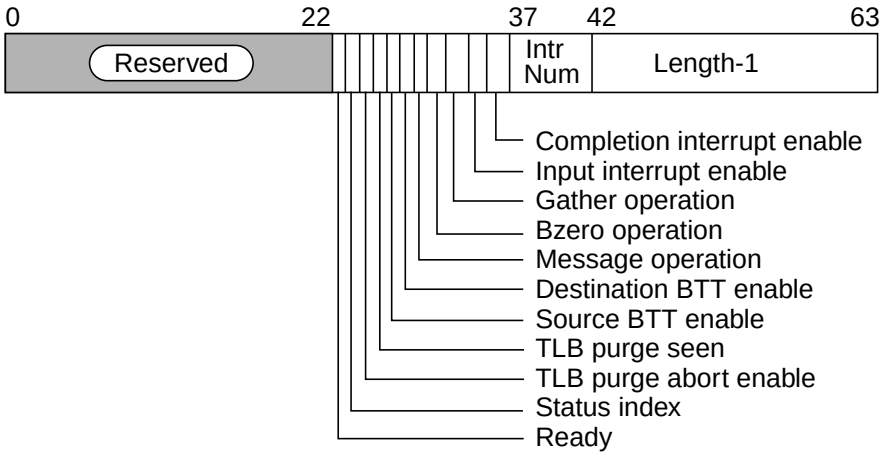
EPAC Input Command registers

Each EPAC has two input command registers that set the modes and lengths of messaging and data copy operations.

The Input Command register can be written when the Ready bit of the CSR is zero and the CSR Operation Context register Armed bit is a one. There are no restrictions for reading this register.

The format of the Input Command register is shown in Figure 23.

Figure 23 EPAC Input Command register format



The bits and fields of the Input Command register are defined as follows:

- *Ready* bit (bit 23)—Indicates that the input registers are ready to perform an operation. Normally, this bit is set by software and cleared by hardware. It should be set by software when the input registers are completely set up for an operation. Hardware clears it when the messaging and copy state machine has accessed all required information from the input registers for the operation. The Ready bit is written by a CSR write access. A CSR read will read the current value. Reset clears the bit.
- *Status index* field (bits 24:25)—Indicates part of the status in the Operation Completion status queue. Reset clears the field.
- *TLB purge abort enable* bit (bit 26)—Enables an operation to be aborted if a TLB purge transaction is detected prior to or during the operation. In system operation, software sets and clears the bit. The operation aborts prior to starting if the TLB purge seen and TLB purge abort enable bits are set at the time the messaging and copy state machine starts the operation. Completion status for an aborted operation is written to the appropriate status queue. The TLB Purge Abort Enable bit is written by a CSR write access and read by a CSR read. Reset clears the bit.
- *TLB purge seen* bit (bit 27)—Indicates that a TLB purge transaction was detected by an EPAC. The bit is set by hardware and cleared by software. It is written by a CSR write. A CSR access reads the current value. Reset clears the bit.
- *Source BTT enable* bit (bit 28)—Indicates the Source Physical Page Frame register contains the address of the BTT used for accessing the source memory region of the operation. The bit is written by a CSR write and read by a CSR read.
- *Destination BTT enable* bit (bit 29)—Indicates the Destination Physical Page Frame register contains the address of the BTT used for accessing the destination memory region of the operation. The bit is written by a CSR write and read by a CSR read.
- *Messaging operation* bit (bit 30)—Forces the messaging and copy state machine to use the messaging mechanism to determine the destination address rather than the destination address of the input register. The bit is written by a CSR write and read by a CSR read.

- *Bzero operation* bit (bit 31)—Forces the messaging and copy state machine to clear the destination memory region rather than copy the source to destination memory region. The bit is written by a CSR write and read by a CSR read.
- *Gather operation* field (bits 32:33)—Specifies the stride used for a gather operation. Currently, this field is disabled and set to zero.
- *Input interrupt enable* bit (bit 34)—Enables an interrupt to the associated processor when the Input Command register is available for reprogramming by software. The most significant five bits of the interrupt number that is sent is specified by this field. The least significant bit of the interrupt number sent is zero. The bit is written by a CSR write and read by a CSR read.
- *Completion interrupt enable* field (bits 35:36)—Enables an interrupt to the associated processor when the messaging and copy state machine completes the operation. The field also determines whether an interrupt is sent when the operation completes with an error if it is sent independently from the status of the operation. The field is written by a CSR write and read by a CSR read.
- *Interrupt number* field (bits 37:41)—Specifies the most significant five bits of the interrupt numbers to be sent to the processor that initiated the request. An interrupt is sent when either of two events occur:
  - When the messaging and copy state machine has completed accessing the input registers
  - When the messaging and copy state machine completes the operation

  The least significant bit of the interrupt number is a zero for the first event and a one for the second. The bit is written by a CSR write and read by a CSR read.
- *Length-1* field (bits 42:63)—Specifies the length of the messaging and copy operation. Messaging operations ignore the least significant 5 bits, forcing the length to be an integer number of memory lines (32-byte increments). Copies, however, can be any byte length. A value of zero in the field copies one byte (one memory line for messaging), and a value of all ones in the field will clear four Mbytes of memory. The bit is written by a CSR write and read by a CSR read.

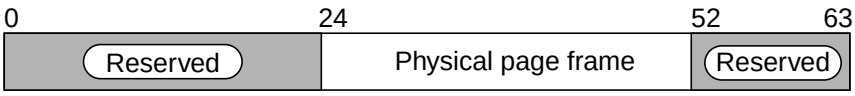
EPAC Source and Destination Physical Page Frame registers

There are two Source/Destination Physical Page Frame registers on each EPAC to specify the source and destination of messaging and data copy operations.

The registers can be written to only when the Input Command CSR Ready bit is zero and the CSR Operation Context register Armed bit is a one. The register can be read at any time.

The format of the Physical Page Frame register is shown in Figure 24.

Figure 24 EPAC Physical Page Frame register definition



The Physical page frame field (bits 24:51) indicates the physical page frame of a 40-bit PA-8200 address. If a BTT is being used, the field specifies the address of the BTT. Otherwise, the field specifies the source or destination page for the copy operation.

For messaging operations, the Destination Physical Page Frame register must be programmed with Node ID = 000 and VR to that of the destination EMAC receiving the message. The Node ID and VR information are written in the normal physical address field positions.

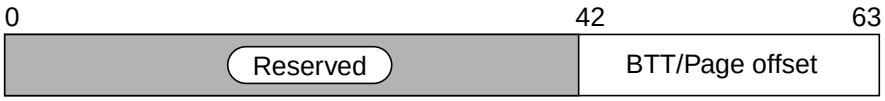
EPAC Source and Destination Offset registers

There are two Source/Destination Offset registers on each EPAC to specify the offset for the source and destination of a message or copy operation.

The registers can be written to only when the Input Command CSR Ready bit is zero and the CSR Operation Context register Armed bit is a one. The register can be read at any time.

The format of the Offset register is shown in Figure 25.

Figure 25 EPAC Source and Destination Offset register definition



The *BTT/Page offset* field (bits 42:63) is used in one of two ways:

- When a BTT is being used, the most significant 10 bits specify the index into the BTT and the least significant 12 bits specify the offset into the selected BTE memory page.
- When a BTT is not being used, the field is used as the offset into a page of memory. The offset within a page can be up to four Mbytes in size for support of larger page sizes.

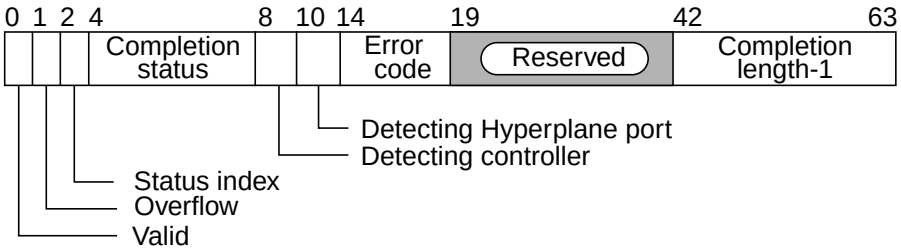
For messaging operations, the Destination Offset register need not be programmed.

EPAC Operation Status Queue registers

There are two Operation Status Queue registers on each EPAC, one for each of the two processors attached to the EPAC. Status can not be inserted in a status queue in the same order the processor sets up the input registers.

The format of the processor 0/1 Status registers is shown in Figure 26.

Figure 26 Operation Status Queue register definition



The bits and fields of the Operation Status Queue register are defined as follows:

- *Valid* bit (bit 0)—Indicates that the Status Queue has valid messaging and copy state machine completion status. The bit is set when the state machine has completed and writes status into the queue. The bit is cleared when the status is read, and no other valid status remains in the status queue. A CSR read access reads the value, and a CSR write has no effect. Reset clears the bit.
- *Overflow* bit (bit 1)—Indicates that a status queue overflow occurred resulting in the loss of status information. The bit is set when a status queue is full and the messaging and copy state machine has completed an operation and its status is destined for that queue. The bit is cleared when the status register is read. A CSR write does not effect the value of the bit. Reset clears the bit.
- *Status index* field (bits 2:3)—Indicates the status index. The two bits are a direct copy of Input Command register Status Index field just before the operation was started.
- *Completion status* (bits 4:7)—Indicates the messaging and copy state machine completion status.

Table 11 Completion status field values

Field values	Completion status
0	Operation completed successfully
1	Data mover detected error
2	Source memory transaction error
3	Destination memory transaction error
4	Source BTE transaction error
5	Destination BTE transaction error
6	Message allocate transaction error
7	Message completion transaction error
8-F	Reserved

- *Detecting controller* (bits 8:9) and *Detecting Xbar port* (bits 10:13) fields—Specify which controller or crossbar port detected the error. This informations is obtained directly from a transaction error response.
- *Error code* field (bits 14:18)—Specifies the type of error that caused the operation to fail.

Table 12 Error code values

Field value	Error code
0	TLB purge aborted operation
1	Insufficient queue space for message
2	Insufficient memory for message
3	Message reception disabled
4	Source BTE translation invalid
5	Destination BTE translation invalid
6	Transaction timed out
7-1F	Reserved

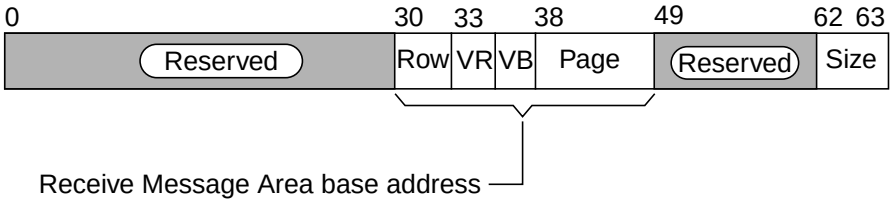
- *Completion length-1* field (bits 42:63)—Indicates the amount remaining to copy when the operation finished. The field is only valid if the operation was aborted with the detection of a TLB purge. The field contains the value of minus one when the operation completed successfully and zero or greater if the operation was aborted. The value restarts an operation when it aborted due to a TLB Purge being detected. A CSR read access reads the value, and a CSR write has no effect.

EMAC Message Reception Area Configuration register

There is one Message Reception Area Configuration register on each EMAC to specify the base address for the region of memory used to receive messages.

The format of the Configuration registers is shown in Figure 27.

Figure 27 EMAC Message Reception Area Configuration register definition



The *Size* field (bits 62:63) specifies the size of the Message Reception Area. Table 13 shows the possible sizes for the Message Reception Area.

Table 13 Message Reception Area size options

Field value	Size
0	32 Kbyte
1	256 Kbyte
2	2 Mbyte
3	16 Mbyte

EMAC Message Reception Area Offset registers

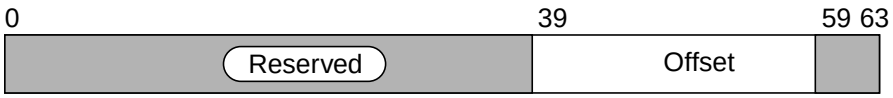
There are two Message Reception Area Offset registers on each EMAC:

- Message Reception Area Available Offset register—Specifies the region of the message reception area available for new messages.
- Message Reception Area Occupied Offset register—Specifies the region presently occupied by messages.

One register specifies the offset into the message reception area where the received messages start and the other specifies where occupied memory starts.

The format of the Message Reception Area Offset registers is shown in Figure 28.

Figure 28 EMAC Message Reception Area Offset register definition



Offset field (bits 39:58) specifies an offset into the message reception area. The register is normally read and written by hardware (to allocate space for new messages). It is read by hardware to check if sufficient available area exists for a new message and written by software to free memory consumed by previously received messages.

Depending on the Size field of the register, some of the most significant bits of the offset field are not used and must be set to zero when written by software. Table 14 shows the bits for each possible size of the message reception area.

Table 14 Offset bits used for each size option

Size option	Bits used as offset
32 Kbyte	10-bits (49:58)
256 Kbyte	13-bits (46:58)
2 Mbyte	16-bits (43:58)
16 Mbyte	19-bits (40:58)

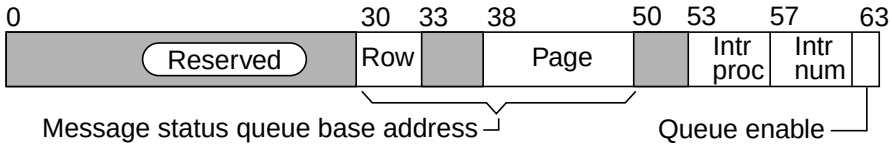
The message reception area is full when the Message Reception Area Available Offset is equal to the Message Reception Area Occupied Offset in the bits specified in Table 14 and the single bit more significant to that specified in the table is different. Bit 39 of the Offset field is never used as an offset to the Message Reception Area, but rather is only used to determine the full status of the Message Reception Area when the size is 16 Mbytes.

EMAC Message Completion Queue Configuration register

Each EMAC has one Message Completion Queue Configuration register that specifies the base address for a region of memory used to write the message completion status.

Figure 29 shows the format of the register. All fields of the register are read by a read access and written by a write access.

Figure 29 EMAC Message Completion Queue Configuration register definition



The bits and fields of the Message Completion Queue Configuration register are defined as follows:

- *Row* (bits 30:32) and Page (bits 38:49) fields—Specify the Message Completion Queue base address. The VB is not part of the base address, because the hardware uses all banks on the EMAC with specified Row and Page values as message completion queue area memory.
- *Interrupt processor* field (bits 53:56)—Specifies which of the 16 processors to interrupt when message completion status is placed in the message completion queue.
- *Interrupt number* field (bits 57:62)—Specifies the interrupt number used to interrupt the destination processor when message completion status is placed in the message completion queue.
- *Queue enable* bit (bit 63)—Enables receiving messages to the associated message reception area. The bit is cleared by reset.

EMAC Message Completion Queue Offset registers

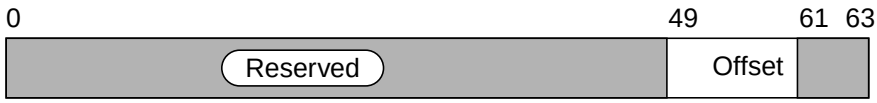
Each EMAC has three Message Completion Queue Offset registers:

- Message Completion Queue Reserve Offset—Specifies the offset into the message completion queue memory area where space has been reserved for message completion status.
- Message Completion Queue Write Offset—Specifies the offset where received message status is written.
- Message Completion Queue Read Offset—Specifies the offset where message completion status is read.

Software must initialize these registers by writing a zero value, but, thereafter, only hardware needs to read or write the registers.

Figure 30 shows the format of the three Message Completion Queue Offset registers.

Figure 30 EMAC Message Completion Queue Offset register definition



The *Offset* field (bits 49:60) specifies an offset into the message completion queue memory area. The most significant bit of the field (bit 49) is not part of the offset, but determines the full or empty status of the queue.

The Message Completion Queue is full when bits 50:60 of the Message Completion Queue Read Offset register are equal to bits 50:60 of the Message Completion Queue Write Offset register and bit 49 of each register is different. The queue is empty when bits 49:60 of each offset register have the same value.

EMAC Message Allocation address

Each EMAC has a message allocation address. This address is special in that it does not have registers associated with it, but rather manipulates other CSRs when accessed. The operation performed is to check that space exists in the message reception area and message completion queue, and, if it does exist, to allocate space in the reception area and reserve an entry in the message completion queue.

The following functionality is performed by an access to this address:

- Checking that the Message Reception Area has been enabled to receive a message.

  This is performed by checking the Queue Enable bit of the Message Completion Queue Configuration register.

- Checking that an entry exists in the Message Completion Queue.

  The information required for the check is the Message Completion Queue Reserved Offset and Message Completion Queue Read Offset registers. The check that is made is that the comparison of the two offsets does not result in queue full.

- Checking that space exists in the message reception area.

  The information needed for this check is the length of the message, the Message Reception Area Available Memory Offset register, and the Message Reception Area Occupied Memory Offset register. The check which is made is that the occupied offset less the available offset is greater than the length of the message.

- Returning status of the unsuccessful allocation attempt if any of the above checks fail.

- Incrementing the Message Reception Area Available Offset register by the length of the message.

- Incrementing the Message Completion Queue Reserved Offset register by one, indicating one less entry available.

EMAC Message Completion Enqueue address

Each EMAC has a Message Completion Enqueue address that is special in that it does not have registers associated with it, but rather other CSRs are manipulated when the address is written to. The operation performed is writing the completion status to a memory-based message completion queue.

The message completion queue should not be full, because any previous access to the Message Allocation register address will have reserved space in the queue for the completion status.

The following functionality is performed by a write to this address:

- Writing of the completion status to the memory-based message completion queue.

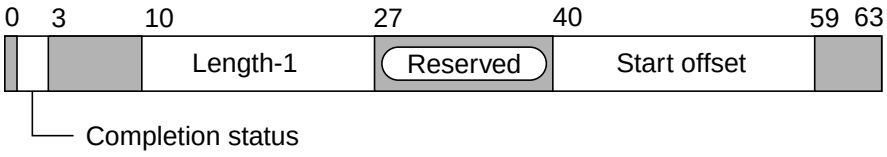
  The memory address to be written is formed by the Row and Page fields of the Message Completion Queue Configuration register and the Offset field of the Message Completion Queue Write Offset register. The data to be written is contained in the write request packet.

- Incrementing by one the Offset field of the Message Completion Queue Write Offset register.

If the Message Completion Queue was empty prior to accessing the Message Completion Enqueue address, the processor specified by the Message Completion Queue Configuration register is interrupted.

Figure 31 shows the format for the request data sent with a write to a Message Completion Enqueue address.

Figure 31 EMAC Message Completion Enqueue definition



The bits and fields of the response data returned from a read to the address are defined as follows:

- *Completion status* field (bits 1:2)—Specifies the completion status of a received message. Table 15 shows the possible completion status field values.

Table 15 Message Completion Status field values

Field value	Completion status
0	Message received successfully
1	Message aborted
2, 3	Reserved

For completion status values 0 and 1, the space for the message was allocated in the Message Reception Area and the memory must be free.

- *Length-1* field (bits 10:26)—Specifies the length of the allocated memory in memory lines (32-byte increments) for the message. A zero value specifies one memory line (32 bytes) and a value of all ones specifies 131,072 memory lines (4 Mbytes).
- *Start Offset* field bits 40:58)—Specifies the offset into the memory reception area to the start of the message.

EMAC Message Completion Dequeue address

Each EMAC has a Message Completion Dequeue address that is special: it does not have registers associated, and it manipulates other CSRs when the address is read. The operation performed is reading the completion status from a memory based message completion queue.

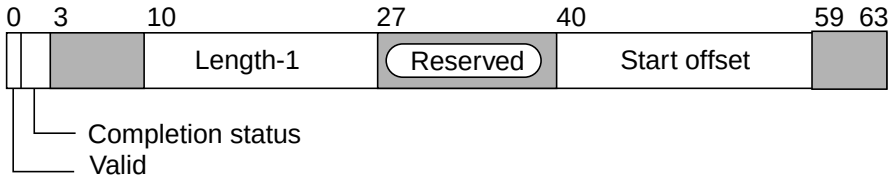
The functionality performed by a read to this address is listed below:

- Returning a response with the valid bit as zero if the Message Completion Queue is empty
- Reading the completion status from the memory-based Message Completion Queue

  The memory address read is formed by using the Row, and Page fields of the Message Completion Queue Configuration register and the Offset field of the Message Completion Queue Read Offset register. The data that is read is returned in the response packet.
- Incrementing by one the Offset field of the Message Completion Queue Read Offset register

Figure 32 shows the format for the response data returned from a read to a Message Completion Dequeue address.

Figure 32 EMAC Message Completion Dequeue definition



The bits and fields of the response data returned from a read to the address are defined as follows:

- *Valid* bit (bit 0)—Indicates the empty status of the Message Completion Queue at the time of the read access.
- *Completion status* field (bits 1:2)—Specifies the completion status of a received message. Table 16 shows the possible completion status field values.

Table 16 Message Completion Status field values

Field value	Completion status
0	Message received successfully
1	Message aborted
2, 3	Reserved

For completion status values 0 and 1, the space for the message allocated in the Message Reception Area and memory must be freed.

- *Length-1* field (bits 10:26)—Specifies the length of the allocated memory in memory lines (32-byte increments) for the message. A zero value specifies one memory line (32 bytes) and a value of all ones specifies 131,072 memory lines (4 Mbytes).
- *Start offset* field (bits 40:58)—Specifies the offset into the memory reception area to the start of the message.

Memory structures

This section describes the memory structures used by the messaging and data copy hardware. The three data structures are:

- Message Reception Area
- Message Completion Queue
- Block Translation Table (BTT)

Message Reception area

The Message Reception area is a preallocated region of memory to which messages can be written. The memory is controlled by hardware that enqueues messages as they are received.

All accesses to message reception areas are through coherent memory accesses. A processor can copy a message out from the Message Reception area directly or by using the data copy hardware.

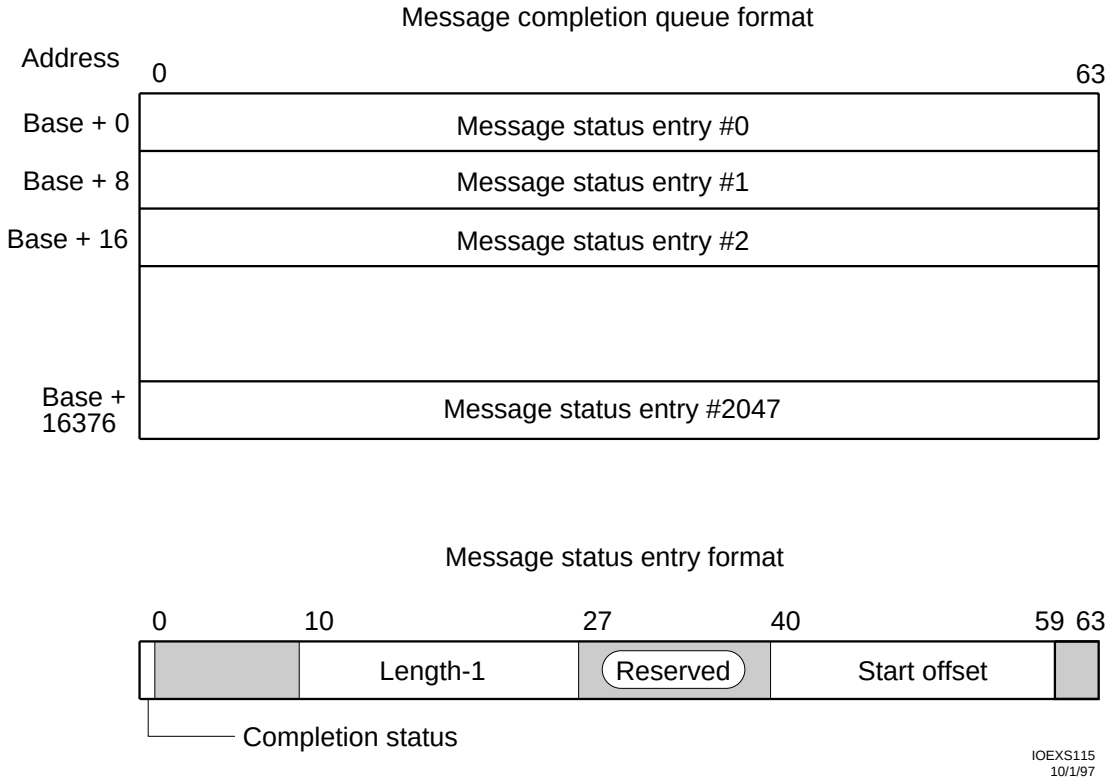
Message Completion Queue area

The Message Completion Queue area holds message completion status until software is ready to process a received message.

The size of the Message Completion Queue area is fixed at 16 Kbytes. Each entry is 8 bytes in size, resulting in 2048 entries per queue. The Message completion queue area resides in memory that is physically connected to the EMAC.

Figure 33 shows the format for a Message Completion Queue and one of its entries.

Figure 33 Message Completion Queue and entry definition



The fields of the message status entry are as follows:

- *Completion status* field (bits 0:1)—Specifies the completion status of a received message. Table 17 shows the possible completion status field values.

Table 17 Message Completion Status field values

Field value	Completion Status
0	Message received successfully
1	Message aborted
2, 3	Reserved

For completion status values 0 and 1, the space for the message allocated in the Message Reception Area and the memory must be freed.

- *Length-1* field (bits 10:26)—Specifies allocated memory in number of memory lines (32-byte increments) for the message. A value of zero specifies one memory line (32 bytes), and a value of all ones specifies 131,072 memory lines (4 Mbytes).
- *Start offset* field (bits 40:58)—Specifies the offset into the Memory Reception Area to the start of the message.

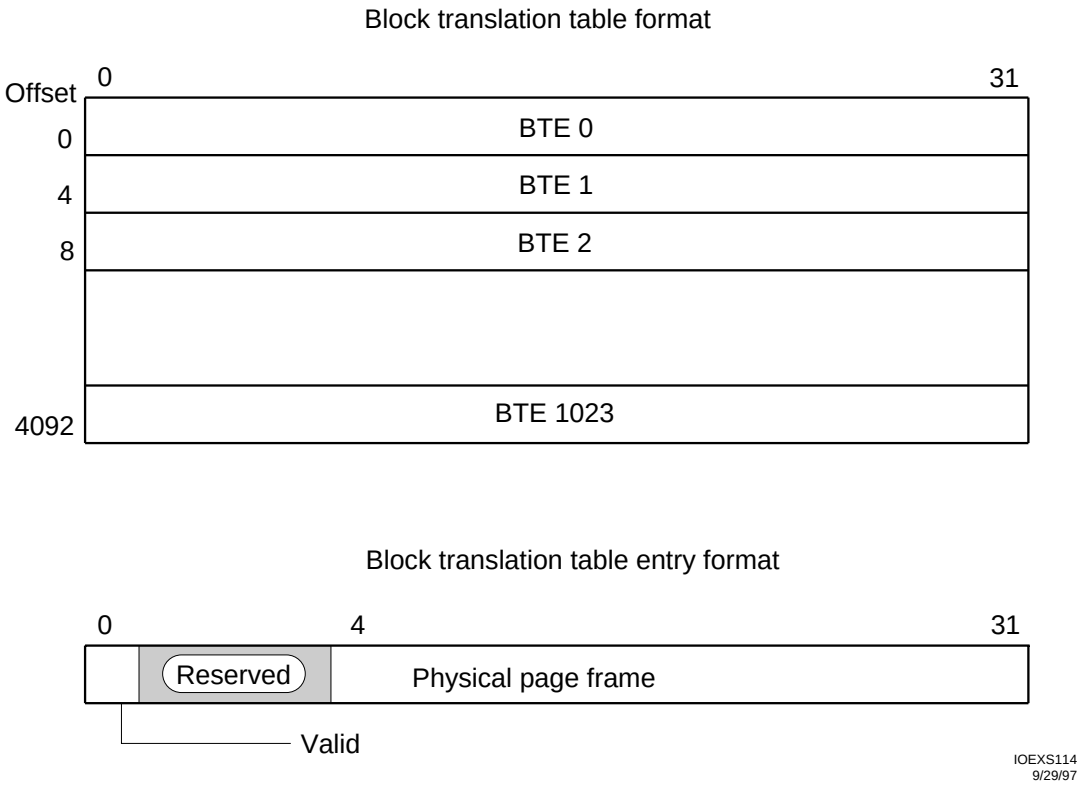
Block Translation Table definition

The BTT provides the I/O system a means to translate from a peripheral's address space to physical memory. It specifies a mapping of contiguous addresses to pages of physical memory. The table is limited to a single page of memory, with each entry being a word (four bytes) in size.

Each entry in the table is called a Block Translation Entry (BTE), and it specifies the page frame for a page of physical memory. A page is 4096 bytes. The BTT specifies a maximum address space of four Mbytes.

Figure 34 shows the format for a BTT and one of its entries.

Figure 34 Block Translation Table and Entry definition



The bits and fields of the BTE are as follows:

- *Valid* bit (bit 0)—Indicates a valid entry. If the messaging and copy state machine needs to use an entry without the *Valid* bit set, the operation is aborted, with completion status indicating the problem.
- *Read/Write* bit (bit 1)—Ignored by the messaging and copy state machine.
- *Physical page frame* field (bits 4:31)—Indicates the page frame for either the source or destination of the operation.

5 Synchronization

Synchronization allows processors executing multiple threads of the same or different processes to share data by locking data structures until the thread requiring the structure has completed. It is accomplished by using a semaphore variable associated with each data structure.

The semaphore variable provides a flag to all processors sharing the data structure. Each processor has access to the semaphore variable, but it can only manipulate the variable atomically. That is, once a processor starts a semaphore operation, it must complete it before any other processor can modify this same data structure.

Coherent semaphore instructions

There are two instructions for semaphore operations in coherent memory:

- Load and Clear Word (LDCW)
- Load and Clear Double (LDCD)

The load implies that the semaphore variable memory line is loaded into the processor data cache.

NOTE V-Class systems require the accelerated cache hint bit be set, because the LDCW instruction without it set is a fetch and clear operation and does not use the data cache. See “Accelerated cache coherence” on page 46.

Noncoherent semaphore operators

The V-Class server supports six semaphore operators not defined in the PA-RISC architecture. These special operators may enhance semaphore operations in some applications, because they operate directly on semaphore variables located in unencacheable memory pages (pages with the U-bit set in the TLB entry; see the section “PA-8200 TLB Entry U-bit” on page 88). They do not accelerate the semaphore into the data cache.

These noncoherent semaphore operators include the single and double versions of the following operations:

- Fetch and Clear
- Fetch and Increment
- Fetch and Decrement

The fetch implies that the semaphore variable goes directly into a processor register. When either the fetch and increment or fetch and decrement instruction reads the variable, the EMAC automatically increments or decrements it.

In addition to these fetch instructions, noncoherent read and write operations are also available to access semaphore variables. If a noncoherent semaphore operator accesses a memory line that is encached by a processor, the semaphore operation will fail, resulting in an error being returned to the processor. All semaphore variables are 16-byte aligned. Semaphore operations to nonaligned variables produce undefined results.

Implementing noncoherent semaphore operations requires the following sequence of instructions using EPAC CSRs:

1. Check write access privilege for the semaphore address.
2. Arm the operation by writing to the EPAC Operation Context register Armed bit. See “EPAC Operation Context registers” on page 57.
3. Write the physical address to the EPAC Operation Address register.
4. Read the Fetch Operation address. The value read is the return value for the semaphore operation.
5. Check the EPAC Operation Context register Triggered bit to make sure the operation was issued. If the Triggered bit is not set, the operation must be restarted. The Armed bit is cleared when the sequence is interrupted by either an external interrupt or a TLB miss.

The other noncoherent semaphore operations and the noncoherent read operation can also use this sequence by using a different Fetch Operation CSR address. The noncoherent write operation is similar to the above sequence, except that the load instruction is replaced with a store instruction with the value to be stored.

As an example, the sequence of instructions for a fetch_and_inc32 is as follows:

```
      PROBEW  fetch_addr             ;Check protection
loop
      LDI     1,%t1
      STD     %t1,(CSR_OP_ARMED)     ;Arm CSR Operations
      LPA     fetch_addr,%t2
      STD     %t2,(CSR_OP_ADDR)      ;Fetch operation addr
      LDW     CSR_FETCH_INC),%t3     ;Issue fetch semaphore
      LDD     (CSR_OP_ARMED),%t4
      BB,*>=  %t4,62,loop            ;check if triggered
```

Barrier synchronization

Not all threads in a multithread process complete at the same time. All threads, however, must typically wait until the last thread finishes. The threads hit a barrier and must be synchronized before continuing.

The barrier synchronization semaphore is a running count of the number of threads that have reached the barrier. The last processor to finish writes a nonzero value to the semaphore address, signalling to the other processors that the threads are synchronized.

An alternate method for barrier synchronization semaphore operations requires a sequence of instructions using EPAC CSRs similar to the sequence discussed in the section “Noncoherent semaphore operators” on page 83.

The following sequence of instructions provides an alternative method for the coherent_inc64() function:

```
    PROBEW  cincd_addr        ;Check protection
loop
    LDI     1,%t1
    STD     %t1,(CSR_OP_CNTX)  ;Arm CSR Operations
    LPA     cincd_addr,%t2
    STD     %t2,(CSR_CINCD)    ;Issue Coh. Inc.
    LDD     (CSR_OP_CNTX),%t4
    BB,*>= %t4,62,loop         ;check if triggered
```

The steps of the sequence are:

1. Check write protection for the operation address.
2. Arm the operation by writing to the CSR Operation Armed register Armed bit.
3. Perform virtual-to-physical address translation.
4. Fetch the Operation address. The value read is the return value for the semaphore operation.
5. Check the EPAC Operation Context register Armed bit to make sure the operation was issued. If the Triggered bit is not set, then the operation must be restarted. The Armed bit is cleared when the sequence is interrupted by either an external interrupt or a TLB miss.

EPAC semaphore addresses

The EPAC has multiple registers and several addresses to implement CSR-based semaphore operations. The two types of registers, the Operations Context register and the Operation Address register, are detailed in the chapter “Data mover.” The addresses are discussed in the following sections.

EPAC Fetch Operation addresses

Each EPAC has six Fetch Operation addresses, three for each processor pair. Reading these addresses triggers one of the following noncoherent fetch semaphore operations:

- Fetch and Increment
- Fetch and Decrement
- Fetch and Clear

If the Armed bit in the Operation Context register is set, an access to one of the Fetch Operation addresses results in a fetch operation to memory. When the Armed bit is set, the address contained in the Fetch Operation Address register becomes the address for the fetch operation. If the Armed bit is not set, the EPAC returns the value zero to the processor rather than the data intended for the fetch operation.

The size field determines whether the operation is word or double word. Any word-aligned address can be used for word operations, and any double-word-aligned address can be used for double-word addresses.

EPAC Noncoherent Read and Write Operation addresses

Each EPAC has two Noncoherent Read Operation addresses, one for each processor pair. Each EPAC also has two Noncoherent Write Operation addresses, one for each processor pair. A read of the noncoherent read address triggers the noncoherent read operation. A write to a noncoherent write address triggers a noncoherent write operation.

If the Armed bit in the Operation Context register is set, the address contained in the Operation Address register becomes the address for the operation. If the Armed bit is not set, the EPAC returns the value zero to the processor rather than the data intended for the noncoherent read operation. For a Noncoherent Write Operation, if the Armed bit is not set, the EPAC drops the noncoherent write.

The access size determines whether the operation is word or double word. Any word-aligned address can be used for word operations, and any double-word-aligned address can be used for double-word addresses.

EPAC Coherent Increment addresses

Each EPAC has two Coherent Increment addresses, one for each processor pair. Writes to these addresses trigger coherent increment operations. If the armed bit in the Operation Context register is set, the address contained in the Operation Address register becomes the address for the fetch operation. If the Armed bit is not set, then the EPAC ignores the write access.

PA-8200 TLB Entry U-bit

Each PA-8200 TLB entry contains a bit that controls whether an access to coherent memory space should accelerate the memory line into its data cache. The V-Class server uses this bit to inhibit noncoherent operations from being moved into data cache.

Table 18 lists the supported semaphore operators and the associated PA-8200 instructions used to issue the operations for the accessed memory page.

Table 18 Semaphore operation instructions

TLB entry U-bit	PA-8200 instruction	Semaphore operation
0 (Coherent)	LDCW	Load and Clear 32-bit
	LDCD	Load and Clear 64-bit
	CINCD	Coherent Increment 64-bit
	LDW	Noncoherent Load 32-bit
	LDD	Noncoherent Load 64-bit
	STW	Noncoherent Store 32-bit
	STD	Noncoherent Store 64-bit

Mixing coherent and noncoherent accesses to a memory line generates an error to the issuing processor.

6 Interrupts

This chapter discusses the interrupt mechanism of the V-Class server.

The *PA-RISC 2.0 Architecture* manual presents a detailed discussion of the interrupt mechanism implemented for the PA-8200 processor, and that material is not presented in this book. Instead, this chapter discusses interrupts registers unique to the V-Class server.

Overview

Interrupts cause process control to be passed to an interrupt handling routine. Upon completion of interrupt processing, a Return From Interrupt (RFI) instruction restores the saved processor state, and the execution proceeds with the interrupted instruction.

When responding to an interrupt, the processor behaves as if a single instruction were fetched and executed, not pipelined. Any interrupt conditions raised by that instruction are handled immediately. If there are none, the next instruction is fetched, and so on.

Faults, traps, interrupts, and checks are different classes of interrupts.

A fault occurs when an instruction requests a legitimate action that cannot be carried out due to a system problem. After the problem has been corrected, the instruction causing the fault executes normally. Faults are synchronous with respect to the instruction stream.

A trap occurs when a function requested by the current instruction can not or should not be carried out. For example, attempting to access a page for which a user does not have privilege causes a trap. Another example is when the user requires system intervention before or after the instruction is executed, such as page reference traps used for debugging. Traps are synchronous with respect to the instruction stream.

An interrupt occurs when an external device requires processor attention. The external device sets a bit in the External Interrupt Request register (EIRR). Interrupts are asynchronous with respect to the instruction stream.

A check occurs when the processor detects a malfunction. The malfunction may or may not be correctable. Checks can be either synchronous or asynchronous with respect to the instruction stream.

Processor interrupts

System interrupts are applied to a processor by writing to its External Interrupt Request Register. The V-Class server interrupts occur from several sources which include:

- Other processors
- I/O subsystem
- Memory subsystem
- Messaging and data copying mechanism
- Time of Century counter loss of synchronization
- Utilities board

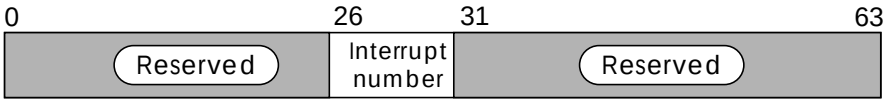
The EIRR is written to using a double word store.

NOTE The fact that the EIRR can only be written to using double word stores is a deviation form the PA-RISC architecture. This is true of all processor CSRs.

All fields of the register are undefined when read.

The format of the EIRR is shown in Figure 35.

Figure 35 PA-8200 External Interrupt Request register definition



The *Interrupt Number* field (bits 26:31) specifies the external interrupt to be set in the EIRR. The value of the interrupt, 0-63, is encoded in the six-bit field.

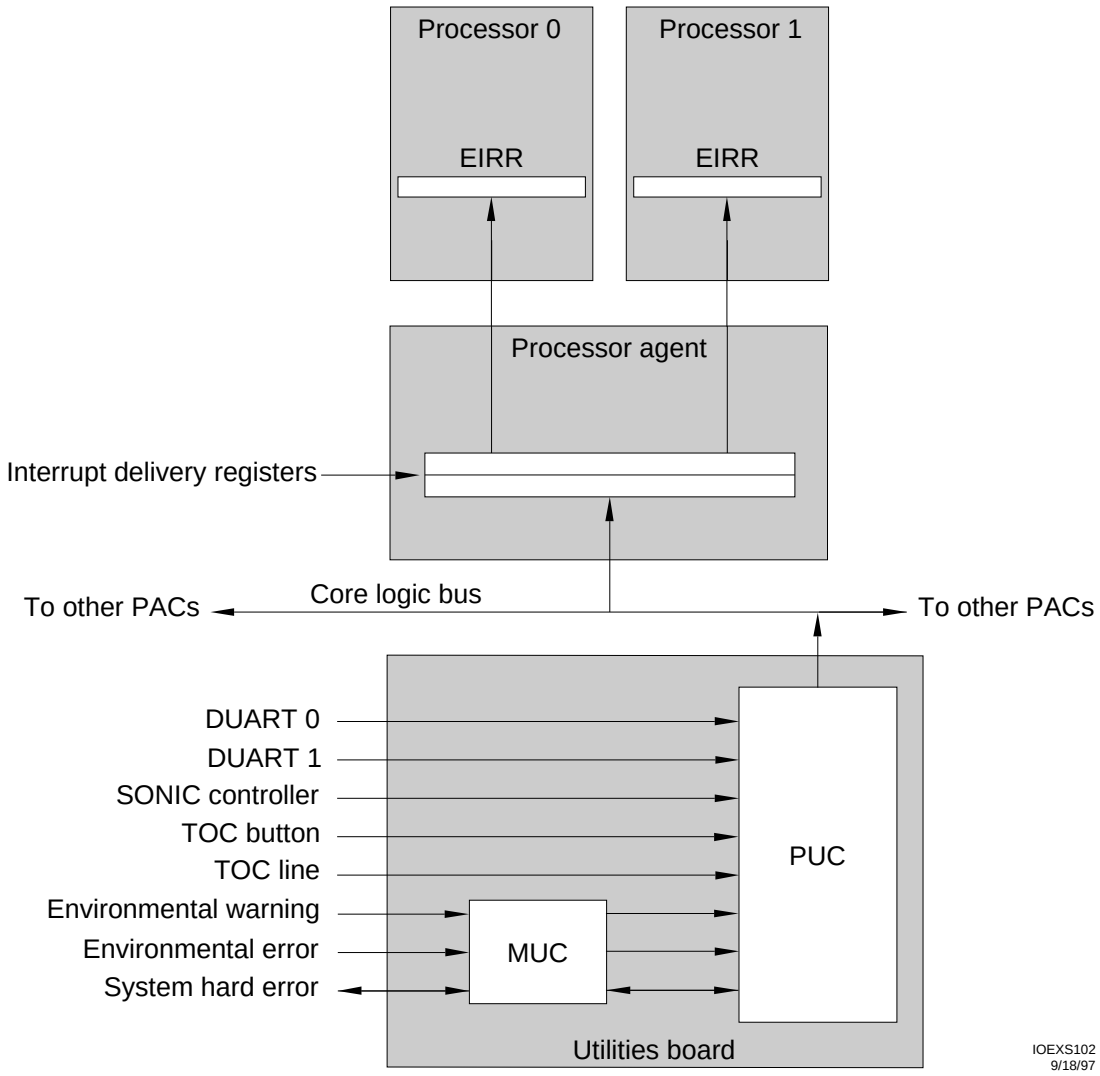
Utilities board interrupts

Almost all interrupts are sent directly to the processor EIRR with the exception of those associated with the core logic bus connected to the Utilities board. The Utilities board collects system environmental interrupts and applies them to the EIRR. The Utilities board handles the following types of interrupts:

- Environmental conditions
- Transfer of control (TOC)
- External communications
- System warnings and failure

Figure 35 shows how these interrupts are presented to the processor.

Figure 36 Core logic interrupt system



The Utilities board provides interrupt information to all EPACs in the system. Each EPAC determines if one of its two processors is enabled to handle the pending interrupt.

The Utilities board accepts eight separate interrupt sources listed in Table 19.

Table 19 Core logic interrupt sources

Core logic interrupt source	Interrupt bit
DUART channel 0	0
DUART channel 1	1
SONIC controller	2
Transfer of control button	3
Transfer of control line (from test station)	4
Environmental warning	5
Environmental error	6
System hard error	7

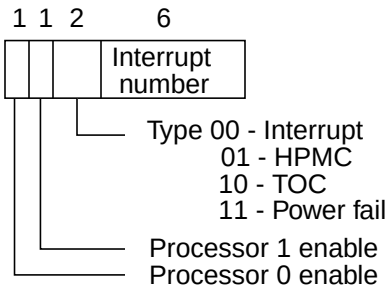
The Utilities board processes interrupts as follows:

- Interrupts are latched into the Interrupt Status register in the EMUC. Interrupts can also be forced into the Force Interrupts register for testing purposes (these are not masked).
- Interrupts are compared to data in the Interrupt Mask register, and, if they are not masked out, are sent across the core logic bus to the Interrupt Delivery register in the EPAC.
- If the EPAC determines that one of its processors has the interrupt enabled, it delivers the interrupt information to the processor by writing to the processor EIRR with the level of the interrupt in bits 26:31 of the 64-bit register, sending a Runway bus transaction to cause an HPMC or writing to the processor I/O Command register to cause a TOC.

EPAC interrupt logic

Each EPAC receives an eight-bit mask from the EPUC (using the core logic bus) that specifies the interrupt sources sent to the processors. Each EPAC has interrupt delivery information for each of the eight possible interrupt sources. Figure 37 shows the interrupt delivery data.

Figure 37 EPAC interrupt delivery information



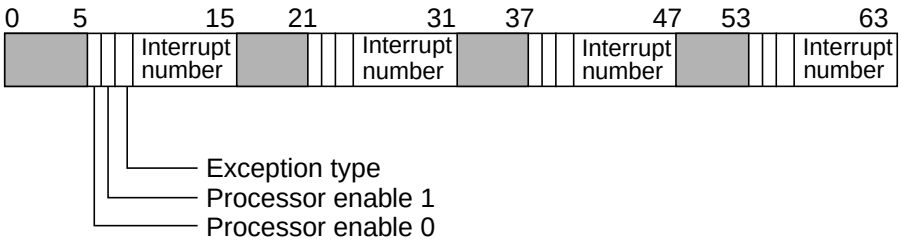
The information contains individual enables for each of the two processors connected to an EPAC, the type of exception, and the interrupt number or an interrupt exception type.

Figure 38 shows where these bits are located in the EPAC Interrupt Delivery register.

EPAC Interrupt Delivery registers

There are two 64-bit Interrupt Delivery registers on each EPAC. Each register specifies the delivery information for four of the eight interrupt sources.

Figure 38 EPAC Interrupt Delivery register definition



The fields and bits of the EPAC interrupt delivery registers are defined as follows:

Processor enable bits—Indicate that the processor is enabled to handle the exception.

Exception type fields—Indicate the type of exception:

- Interrupt
- HPMC
- TOC loss of synchronization
- Power failure

Interrupt number fields—Indicate the interrupt source to the delivery registers.

The Utilities board interrupts map to the core logic interrupt delivery registers as shown in Table 20. All fields are written to by a CSR write and read using a CSR read. Reset has no effect on the register.

Table 20 Core logic interrupt delivery registers

Utilities board interrupt source	Register and bits
DUART channel 0	Register 0, bits 6:15
DUART channel 1	Register 0, bits 22:31
SONIC controller	Register 0, bits 38:47
Transfer of control button	Register 0, bits 54:63
Transfer of control line (from test station)	Register 1, bits 6:15
Environmental warning	Register 1, bits 22:31
Environmental error	Register 1, bits 38:47
System hard error	Register 1, bits 54:63

EPUC interrupt logic

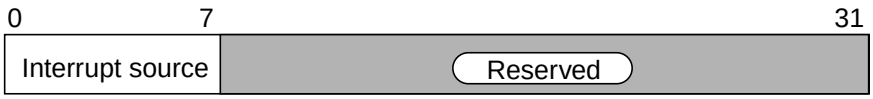
The following EPUC interrupt registers comprise the EPUC interrupt logic:

- Interrupt Status register
- Interrupt Mask register
- Interrupt Force register

EPUC Interrupt Status register

The EPUC contains one Interrupt Status register. The register maintains the status of the pending Utilities board interrupts given in Table 20. Figure 39 shows the definition of the register.

Figure 39 EPUC Interrupt Status register definition



The *Interrupt source* field (bits 0:7) indicates the source of the EPUC interrupt. The bits are set when the EPUC detects an active input interrupt signal. The contents of the register are read by a CSR read operation. Each bit set in the data of a CSR write operation clears the associated bit of the status register, and a reset clears the register.

Table 21 shows the bit field assigned to each core logic interrupt source.

Table 21 **EPUC Interrupt register field definitions**

Register bit	Interrupt source
0	DUART channel 0
1	DUART channel 1
2	SONIC controller
3	Transfer of control button
4	Transfer of control line (from test station)
5	Environmental warning
6	Environmental error
7	System hard error

Each individual bit of the Interrupt Status register can be cleared without affecting the other bits, even when the CSR is receiving an interrupt. For all bits except the transfer of control button, clearing a bit has precedence over setting it. This means that if an input interrupt is still asserted when the bit is cleared, the status bit is set on the following cycle, and a new interrupt is sent to each EPAC. The transfer of control button interrupt is edge-level sensitive, and the other interrupts are level sensitive.

EPUC Interrupt Mask register

The EPUC contains one Interrupt Mask register. The register enables sending the pending interrupts to the EPAC. It provides the ability to mask out any of the eight interrupt sources. Figure 40 shows the definition of the register.

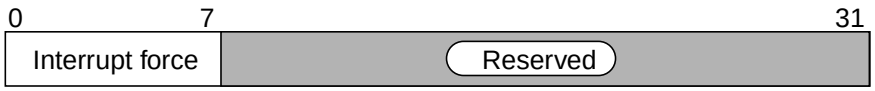
Figure 40 **EPUC Interrupt Enable register definition**

The *Interrupt mask* field (bits 0:7) indicates interrupts are masked. The contents of the register are read by a CSR read operation and written by a CSR write operation. A reset clears the register.

EPUC Interrupt Force register

The EPUC contains one Interrupt Force register. The register allows software to force an interrupt on any of the eight interrupts. Figure 41 shows the definition of the register.

Figure 41 EPUC Interrupt Force register definition



The *Interrupt force* field (bits 0:7) indicates the interrupt(s) being forced. The contents of the register are read by a CSR read operation and written by a CSR write operation. Setting a bit forces an interrupt, regardless if it is enabled of not. A reset clears the register. Table 21 shows the interrupts assigned to each core logic interrupt force bit.

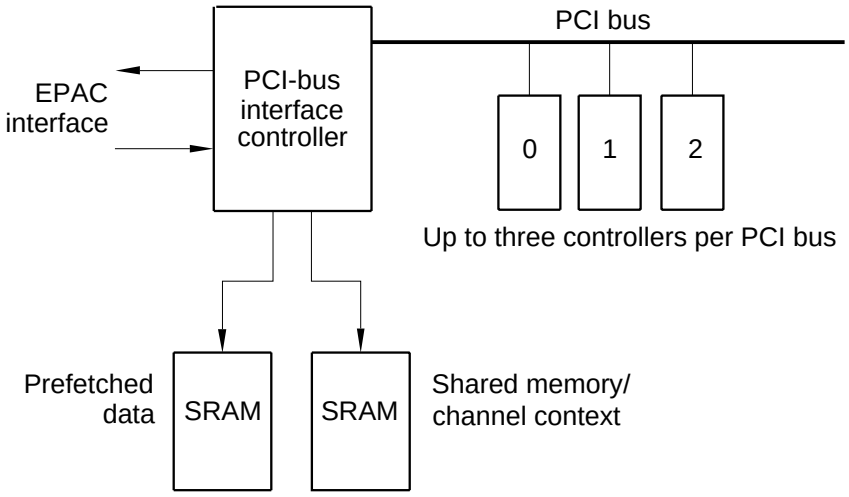
7 I/O subsystem

The I/O subsystem connects the system to its peripheral devices using the industry standard 32-bit peripheral component interface (PCI) bus. A fully configured system provides up to eight PCI buses, one for each EPAC. Each bus supports three controllers for a maximum of 24 PCI controllers for the system.

Overview

The I/O subsystem transfers data coherently to and from the system main memory, eliminating the need for flushing the processor caches. Figure 42 shows a block diagram of the I/O subsystem based on the PCI-bus Interface Controller (EPIC).

Figure 42 **I/O system block diagram**



The EPIC provides memory-mapped access from the processor to the I/O controllers and allows external devices to transfer data into and out of system memory. There is one EPIC per EPAC. Each EPIC has a pair of unidirectional links to the associated EPAC. Each EPIC has two physical SRAM banks, one for EPIC data prefetch and one for PCI controller-shared memory.

Logical I/O channel

The EPIC uses the concept of a logical I/O channel to translate PCI addresses and prefetch system coherent memory. A logical channel defines a pipe between four Mbytes of PCI memory space to four Mbytes of system coherent memory.

Each channel has a distinct address mapping between the PCI bus address space and the system main memory. It also has a buffer for storing prefetched data during read data transfers. The buffer hides PCI start-up latencies associated with read data transfers.

The logical I/O channel also has a posted write buffer for collecting 32-byte data cache lines before flushing them to system coherent memory. Figure 43 depicts the logical I/O channel concept, and Figure 44 shows the PCI bus command and address format.

Figure 43 Logical I/O channel model

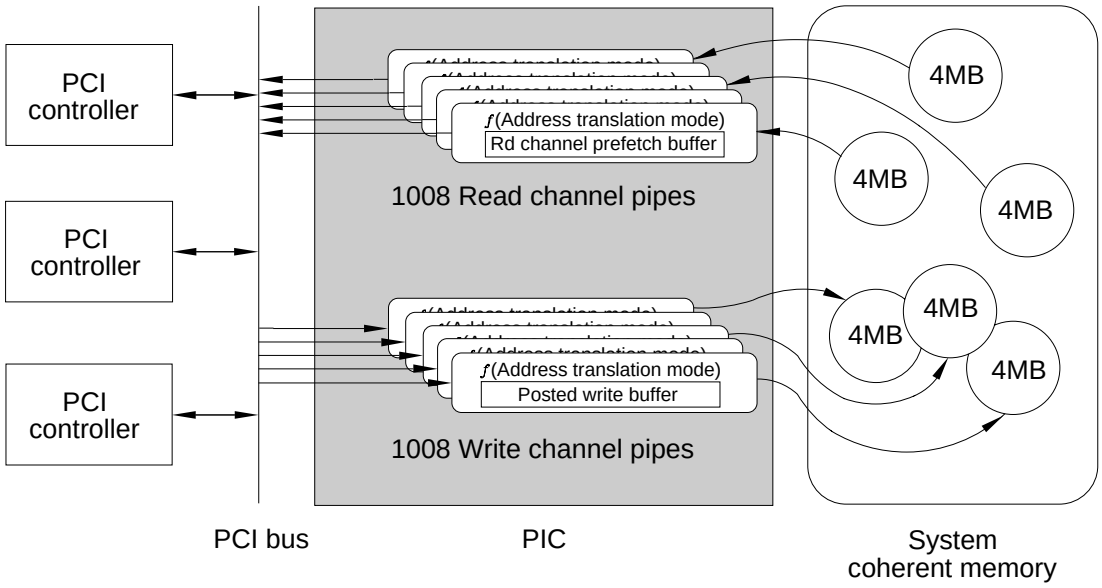
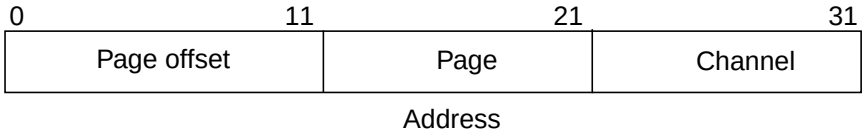


Figure 44 **PCI bus command and address**



The 10 most significant bits of the PCI address define the logical channel number, providing a total of 1,024 logical channels. Channels 1008-1023 are reserved, leaving a maximum of 1,008 read channels and 1,008 write channels. The 22-bit channel offset gives each channel a four-Mbyte data space. Consecutive channels may be chained to allow transfers larger than four Mbytes.

NOTE

Each channel can be used for one or more DMA transfers on a controller. Best performance is usually realized, however, with a single I/O transfer per channel. A channel can not be used by multiple controllers at the same time.

Channel initialization

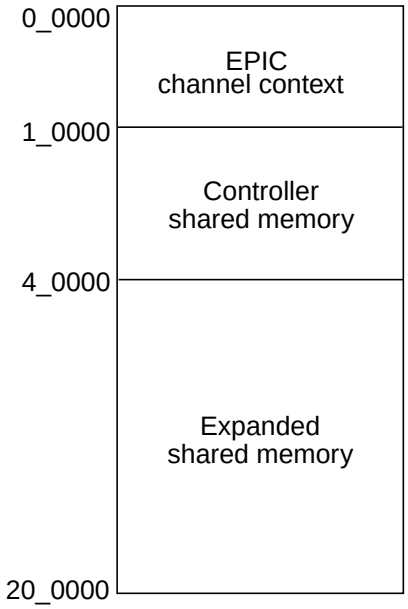
Before a processor initializes an I/O operation, it must set up a channel for the appropriate controller by writing to the EPIC Channel Builder register.

The build consists of a single write to the Channel Builder register. See the section “EPIC Channel Builder register” on page 124. The EPIC initializes all the external SRAM channel context state and prefetches any needed data and TLB entries.

Channel context and shared memory SRAM

The EPIC maintains both channel context and shared memory in its external Channel Context SRAM (CCSRAM). The channel context space reserves 64 Kbytes from the base of the SRAM, and shared memory for both controller and expanded is available for the remainder. The EPIC supports up from 256 Kbytes to 2 Mbytes of external CCSRAM. See Figure 45.

Figure 45 CCSRAM Layout



Channel context

The channel context portion of the SRAM contains information to determine how to perform the DMA transfer between PCI and system memory. Channel context is mapped into both PCI memory space and processor I/O space. The channel context region, however, is only directly accessed for diagnostic use. The processor programs channel context state through the EPIC Channel Builder register.

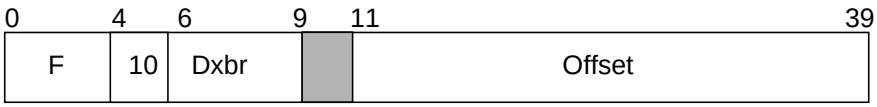
Shared memory

The EPIC provides a locally shared memory region in the CCSRAM for all status and control structures that support the PCI controllers. This SRAM is not coherent with main memory. It is *visible*, however, from both PCI memory space and processor I/O space.

Host-to-PCI address translation

The 40-bit system address map, shown in Figure 46, reserves 16 Gbytes from F8 0000 0000 to FB FFFF FFFF for host access to PCI devices.

Figure 46 I/O address space format



The fields for the I/O address space are defined as follows:

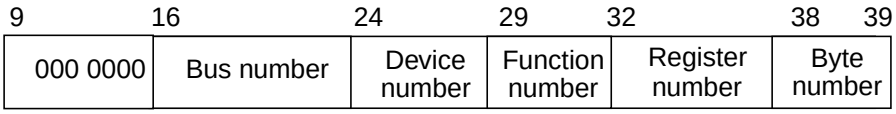
- *Dxbr* field bits (6:9)—Specifies one of eight EPACs
- *Offset* field bits (11:39)—Specifies 29-bit EPIC mapping.

PCI configuration space

The PCI specification establishes three PCI address spaces: configuration, I/O, and memory. Dedicated read and write commands select a particular space for a PCI bus operation.

The PCI configuration space contains a set of configuration registers that must be implemented by all bus targets except host bridges. The configuration registers allow the EPIC to set up PCI I/O and memory space requirements in the system address map. PCI configuration space is 16 Mbytes (24 bits). Figure 46 shows the PCI configuration address format.

Figure 47 I/O PCI configuration space format



The fields for the I/O configuration space format are defined as follows:

- *Bus number* field (bits 16:23)—Indicates PCI bus number. Bus 0 is the bus directly attached to the EPIC. Any other PCI buses must be assigned Bus numbers 1 to 255 during the software probe.
- *Device number* field (bits 24:28)—Specifies the device on one PCI bus segment. Bus 0 only supports Device 0 through Device 3.
- *Function number* field (bits 29:31)—Specifies the function on a PCI device.
- *Register number* field (bits 32:37)—Specifies the register within a PCI function.
- *Byte number* field (bits 38:39)—Provides the byte address. This field and the packet size code establish the PCI byte enables during the access. Accesses must be aligned to their natural size. The EPIC does not support 64-bit double-word accesses to PCI.

PCI I/O and memory space

PCI I/O and PCI memory space allow host access to device-specific CSRs. Target implementation of either space is optional. However, if a device implements either space, it must also implement a corresponding base address register in PCI configuration space to allow consistent address mapping.

PCI I/O and PCI memory space may each be as large as four Gbytes. PCI I/O space uses a full byte address, so the EPIC combines the least significant bits of the system address with the packet size code to create the PCI byte address and the PCI byte enables. PCI memory space uses four byte-aligned addresses; smaller entities are addressed by bus byte enables.

I/O space-to-PCI map

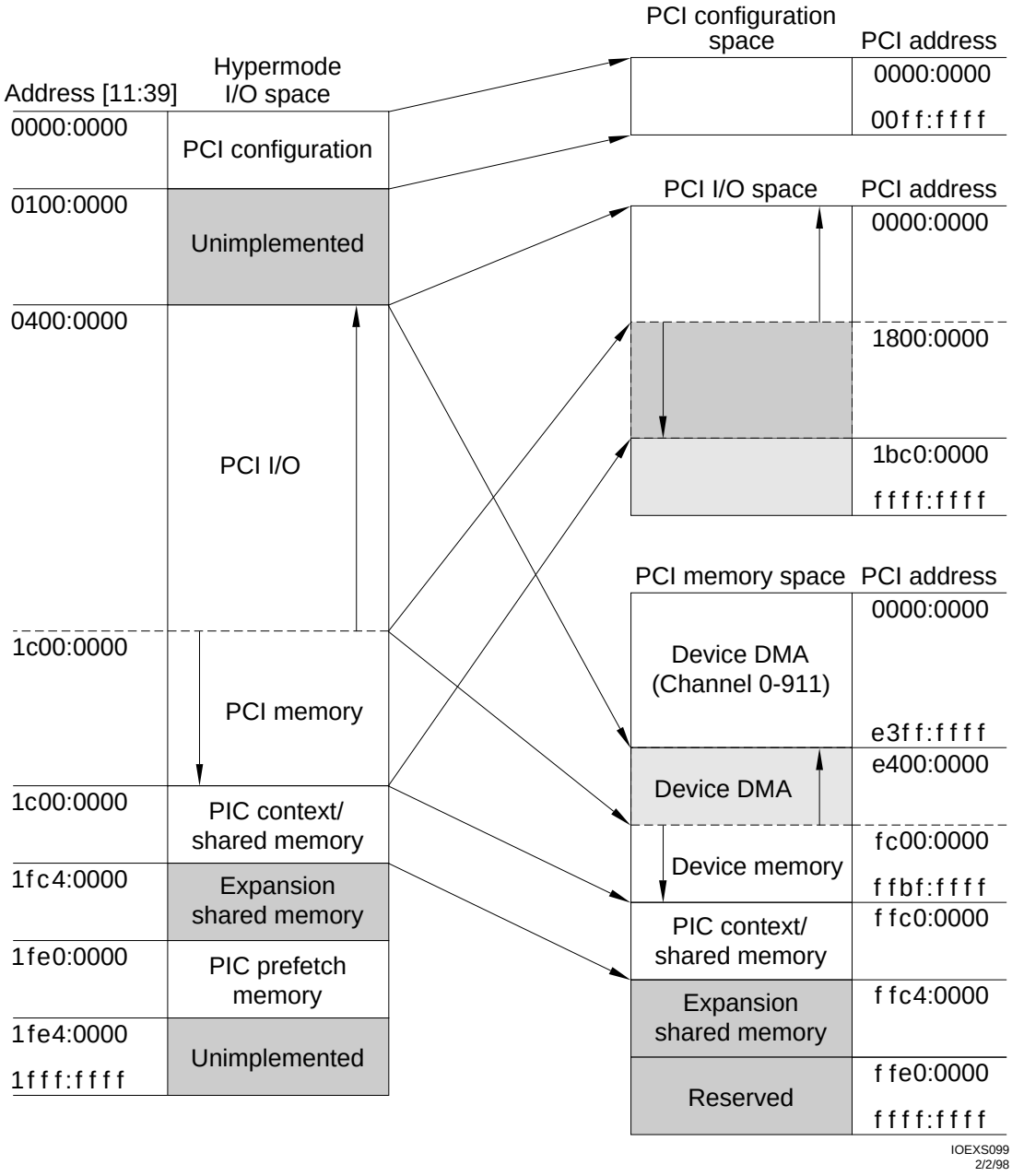
As shown in Figure 48, the EPIC maps its partition of I/O space into the three PCI spaces. It also reserves an area for diagnostic windows into the external EPIC context/shared memory and external EPIC prefetch memory.

The PCI defines eight Gbytes of I/O and memory space, but the EPIC only has 0.5 Gbyte of space in which to operate. Therefore, the address map is necessarily sparse. Only the PCI configuration space maps on a one-to-one basis.

The EPIC can generate PCI addresses, increasing from 0000 0000 in PCI I/O space and decreasing downward from FFBF FFFF in PCI memory space. The allocation boundary between I/O and memory space is programmable in 64-Mbyte increments and can range from no I/O space and all memory space to no memory space and all I/O space.

Maximizing the PCI I/O space also maximizes the number of available PCI DMA channels, while increasing the PCI memory space comes at the cost of 16 PCI DMA I/O channels per 64-Mbyte increment.

Figure 48 I/O space to PCI space mapping



PCI-to-host memory address translation

Since most PCI controllers generate a 32-bit address, they are capable of addressing up to four Gbytes. The V-Class server can have more than this amount. Therefore, it provides for translating the 32-bit addresses to system addresses. There are two types of address translation: physical and logical. An Address Translation Enable bit (ATE) for each channel determines the address translation between PCI and system coherent memory.

In the physical translation mode, data is fetched directly from a four-Mbyte buffer in system main memory.

Logical address translation implies that the translation process uses an intermediate step to derive the system address. The process uses translation tables in system memory for data transfers.

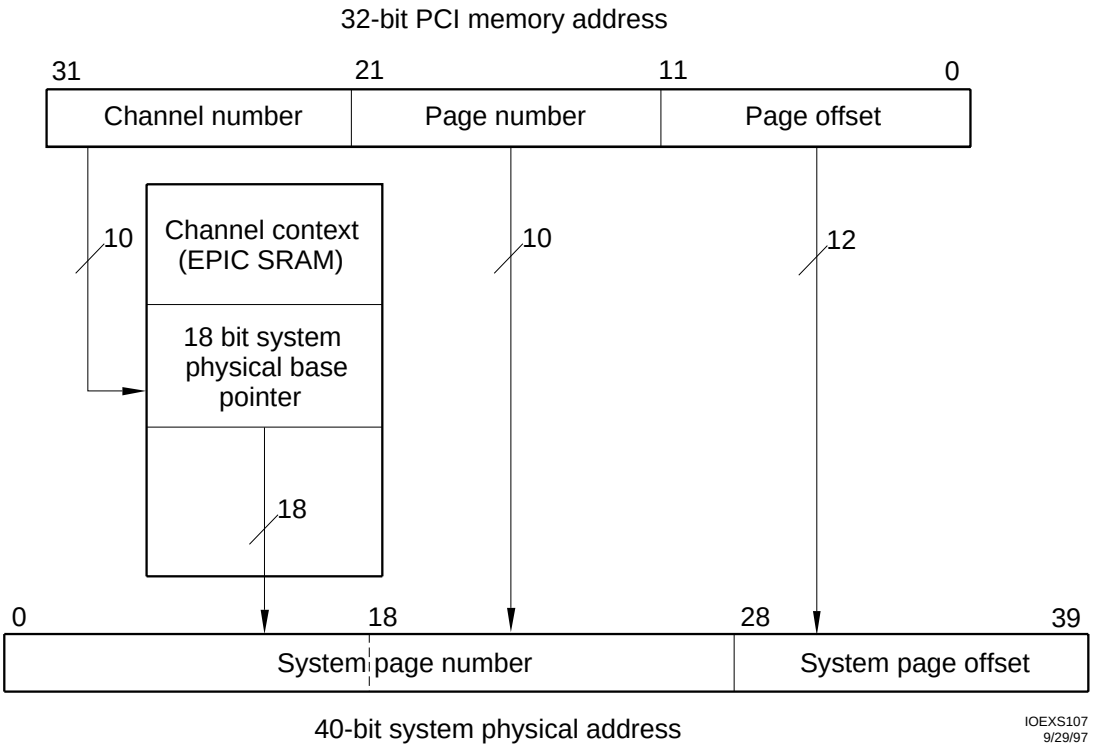
Most modern I/O controllers use part of the host memory for storing control and status blocks. Typically, these are accessed using word accesses over the PCI bus. Since the main memory access latency is relatively large, part of the channel context SRAM is used for storing the control and status structures.

By addressing logical channel 1023, a controller accesses the entire SRAM.

Physical address translation

The simplest translation mode is the physical translation mode. In this mode, the four-Mbyte PCI channel directly maps into a four-Mbyte, physically contiguous block of system memory. The 22-bit PCI channel offset is combined directly with the 18-bit channel physical base pointer to generate the 40-bit system address.

Figure 49 Physical mode address translation



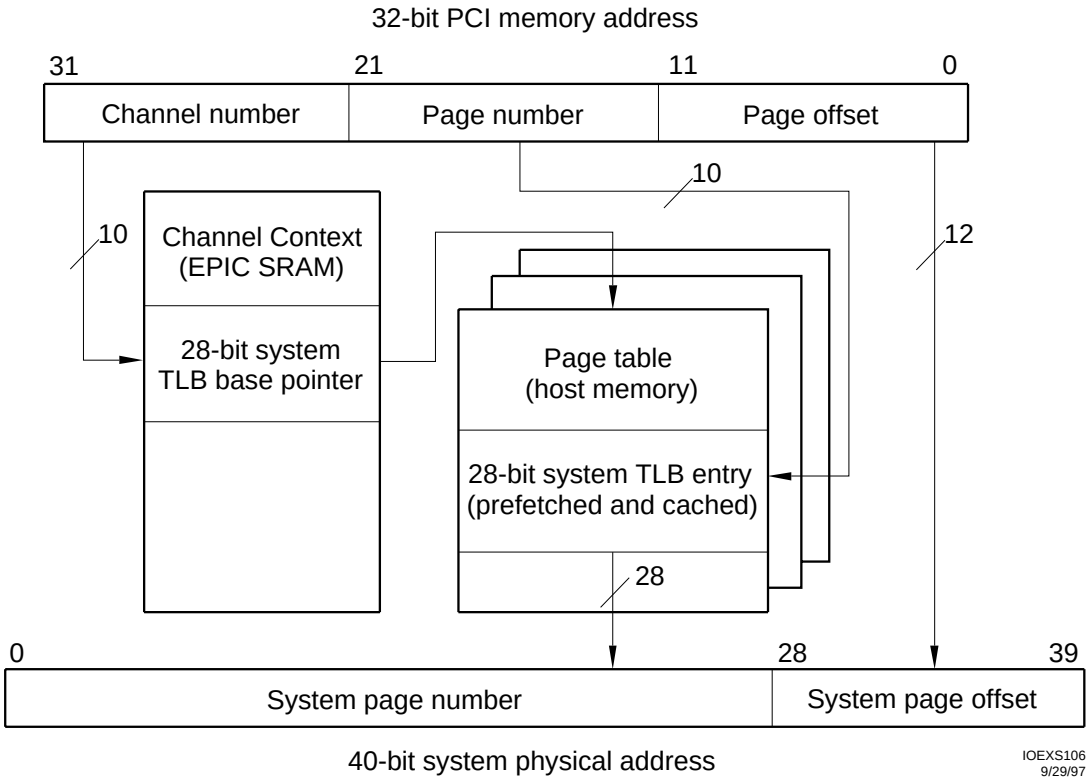
Some I/O transfers, specifically remote receive transfers with many small I/O streams, need to be handled in a nondeterministic order. If each transfer were located in its own channel, software could run out of channels. If the transfers are packed into a single logical channel, the TLB miss overhead when switching streams would then limit the controllers throughput.

Software can pack remote receive buffers into a single physical channel and reduce the number of channels used, reduce the number of channel swaps, and eliminate TLB miss latencies.

Logical address translation

The more common way to map the 32-bit PCI address into the 40-bit system address is using a logical translation mode channel. For logical translations, a translation table is used to generate the 40-bit system address from the 32-bit PCI address.

Figure 50 Logical mode address translation



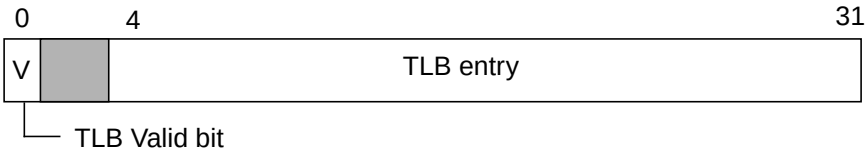
The logical address translation mode is based on a translation lookaside buffer much like the processor TLB. The translation table converts the PCI bus addresses into system addresses on a page (four-Kbyte) basis, and therefore, are aligned on page boundaries. A TLB base pointer points to the page of TLB entries in system memory. The PCI page

number indexes this table, pointing to a system page number. This system page number and the PCI page offset are combined to generate the 40-bit system address.

I/O TLB entry format

The I/O TLB entries in system coherent memory are the same as those used by the data mover. They are not the same as the processor TLB entries.

Figure 51 I/O TLB entry format



An I/O page table consists of 1,024 TLB entries. Each 28-bit TLB entry points to a four-Kbyte page of system coherent memory. Therefore, the table consumes four Kbytes of system coherent memory. The channel TLB base pointer points to its channel page table. The PCI page number indexes the page table to address the needed TLB entry. See Figure 50.

PCI memory read transfers

To handle long and variant system memory latencies, the EPIC uses several different prefetch techniques in combination to ensure that the data needed by a controller is available at the time it is needed. These techniques include:

- Channel prefetch/refetch
- Device consumption-based prefetch
- Device stall prefetch

These techniques allow the EPIC to:

- Compensate for start-up latencies from memory
- Maintain a minimal prefetch depth that matches the memory latency needed for prefetch data to the controller consumption rate of that data
- Increase prefetch depth dynamically, as needed, to account for larger latencies than the current depth of the prefetch buffer can hide

Read data is coherent at the point that the request is satisfied in system memory. I/O transfers are not included, however, in system memory sharing lists. Therefore, if the data is modified later, the EPIC prefetched data will be stale. This fact dictates that direct memory accesses (DMA) from system memory be used only for buffered data that is defined prior to the EPIC issuing any prefetches for that data. To purge prefetched data from the EPIC prefetch buffers, the channel must be either reinitialized or rebuilt through the Channel Builder register.

To accommodate these prefetch techniques, the EPIC provides two types of data prefetch storage:

- Channel Prefetch space
- Device Prefetch space

The channel prefetch space hides start-up latencies, and the device prefetch space maintains the streaming data. When a PCI transfer starts, data is supplied from the smaller channel prefetch buffer. Once that data from this buffer is exhausted, the data is pulled from the larger device prefetch buffer.

Channel prefetch space

Channel prefetch space stores channel prefetch data. This space hides the typical start-up latency for system accesses when a controller switches from one channel to another.

There is one channel prefetch buffer per channel for a total of 1,008 channel prefetch buffers. The amount of storage space needed to cover the start-up latency for system accesses determines the depth of each channel prefetch buffer. The depth of the buffer is sized with the following formula:

Channel prefetch depth = PCI bandwidth * Local memory latency

Device prefetch space

Device prefetch space stores the prefetch data of a streaming device. It buffers a single controller stream of data from memory. The depth of the buffer is sized to hide latencies with minimal stalls on the PCI bus.

There is one device prefetch buffer per controller. The depth of the buffer is sized with the following formula:

Device prefetch depth = PCI bandwidth * Remote memory latency

Channel prefetch/refetch modes

For small transfer high bandwidth controllers, the start up latency dictates the effectiveness of the controller to move data. The start up latency is the time from which a controller provides the EPIC a new address stream to the time the EPIC provides the first data word.

The EPIC provides a Channel prefetch enable (P) bit to hide controller start-up memory latencies. When enabled (P is set to 1), the EPIC Channel Builder register prefetches data at channel initialization time. The prefetched data is stored locally in the channel space of the EPIC external SRAM. Therefore, when a controller presents the EPIC with an address mapping into this channel, the data is already local to the EPIC, reducing the latency to first data word.

A controller that uses time-multiplexing on its read streams (for example, an ATM controller) can also be programmed with a Channel Refetch (R) bit. When this bit is enabled (R set to 1) and one channel is swapped for another, the EPIC first refetches data into the channel

prefetch buffer, starting where the controller left off. This guarantees that when the controller comes back to this address stream, the next needed data is available in channel prefetch space.

Device consumption-based prefetch

Each EPIC can be connected to controllers with different bandwidth requirements. The EPIC must ensure that each controller has fair access to the system memory.

The EPIC consumption-based prefetch algorithm keeps the prefetch request rate matched to the amount of data that a controller is consuming from the EPIC. Each time a line of data is transferred across the PCI interface to a controller, a prefetch is scheduled for that EPIC device prefetch buffer. This also ensures that the depth of the prefetch buffer is maintained at the minimal level that satisfies the consumption rate of the controllers, keeping the EPIC from over prefetching for a particular controller.

Stall prefetch

Occasionally a controller needs data that is not available in the EPIC device prefetch buffer (for example, when the controller address stream jumps outside the depth of the device prefetch buffer). In this case, the stall prefetch mechanism causes the EPIC to issue a constant stream of data prefetches at a programmable interval until the critical line returns to the EPIC. Once prefetch data is available, the prefetch algorithm reverts to the consumption-based prefetch algorithm.

PCI memory write transfers

The EPIC has one independent write buffer per PCI controller. In order to minimize the write traffic to memory, a write buffer accumulates sequential bytes into a cache line of data prior to sending it to system memory. Any of the following events can cause the EPIC to flush this write buffer to memory:

- The controller writes the last byte of a cache line.
- The controller writes a noncontiguous byte stream (a jump).
- A synchronization event forces a write pipe flush.

When the controller write buffer accumulates a cache line of data, a `WritePurge` operation flushes the line of data to memory. When the memory subsystem receives the data, it purges this line from all processors.

When a partial line needs to be flushed to memory, however, a `WritePurge` can not be used, since the current cache line in memory must be merged with the partial cache line of the EPIC. The EPIC provides a Write Purge Partial (W) mode bit per channel that defines how the EPIC should perform this partial cache line merging.

Write purge partial disabled

If the Write Purge Partial bit is cleared, the nonwritten portion of the cache line in memory must be maintained coherently. Therefore, the EPIC must perform a `Dflush_Alloc/Write_Mask` flow. The EPIC first issues a `Dflush_Alloc` to flush the line back to memory, locking down the line. When `Dflush_Alloc` is complete, the EPIC issues the `Write_Mask` operation. This operation writes the data to memory with a mask to allow the memory subsystem to merge the two lines together and release the line in memory.

Write_Purge_Partial enabled

If the Write_Purge_Partial bit is set, there is no guarantee that the nonwritten portion of the cache line in memory is coherently maintained. Setting this bit provides accelerated partial line transfers to system coherent memory, making this mode suitable for transfers like those to kernel buffers but not suitable for I/O transfers directly to user space.

The Write_Purge_Partial provides a mask that defines the data to be written to memory. The remaining bytes of the cache line come from what is currently in memory. When this data is received, the memory subsystem purges any users of the cache line.

I/O subsystem CSRs

The EPIC is controlled by CSRs. All CSRs are 64-bit aligned and may only be accessed using noncoherent Read Short and Write Short packets. The EPIC registers include:

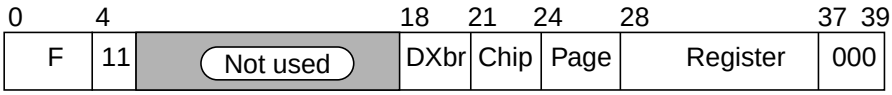
- EPIC Chip Configuration
- EPIC PCI Master Configuration
- EPIC PCI Master Status
- EPIC Channel Builder
- EPIC Interrupt Configuration
- EPIC Interrupt Source
- EPIC Interrupt Enable
- PCI Slot Configuration
- PCI Slot Status
- PCI Slot Interrupt
- PCI Slot Synchronization

EPIC CSR address decoding

The EPIC CSR address decoder looks at system address bits [4:5] to determine the target address space (I/O or EPIC CSRs) and bits [24:39] to index the CSR space. Accesses to unimplemented EPIC CSR space return error responses to the requestor. Reserved addresses and bits ignore writes and return zeros on reads.

The EPIC CSRs may be accessed by addressing, the mapping of which is shown in Figure 52.

Figure 52 EPIC CSR 40-bit address format



The bits and fields of the EPIC CSR space address are as follows:

- *DXbr* field (bits 18:19)—Specifies which of the eight cross bar ports the request is to be routed.
- *Chip* field (bits 21:23)—Routes the packet to the appropriate chip at a crossbar port.
- *Page* field (bits 28:36)—Separates groups of CSRs into similar usage spaces.

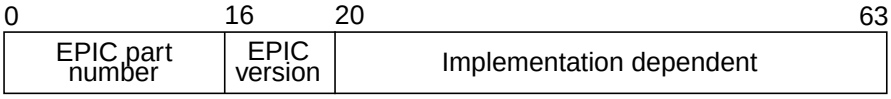
EPIC CSR definition

This section describes the EPIC CSRs.

EPIC Chip Configuration register

The EPIC contains one EPIC Chip Configuration register on each EPIC. It specifies configuration information.

Figure 53 EPIC Chip Configuration register definition



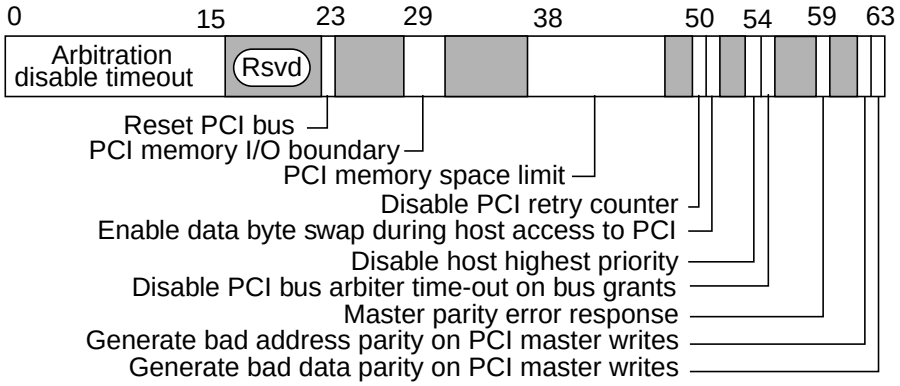
The fields of the EPIC Chip Configuration register are defined as follows:

- *EPIC part number* field (bits 0:15)—Specifies the part number for the EPIC chip. A write is ignored and a read returns the hardwired value.
- *EPIC version code* field (bits 16:19)—Specifies the version of the EPIC chip. A write is ignored and a read returns the hardware value.
- *Implementation dependent* field (bits 20:63)—Specifies implementation dependent information. The value in this field should not be modified during normal use.

PCI Master Configuration register

There is one PCI Master Configuration register on each EPIC that provides configuration information about the PCI Master interface. The format of the register is shown in Figure 54. All reserved fields are read as zero, and writes are ignored. All implemented fields are read with the last value written.

Figure 54 **PCI Master Configuration register definition**



The fields and bits in the EPIC PCI Master Configuration register are defined as follows:

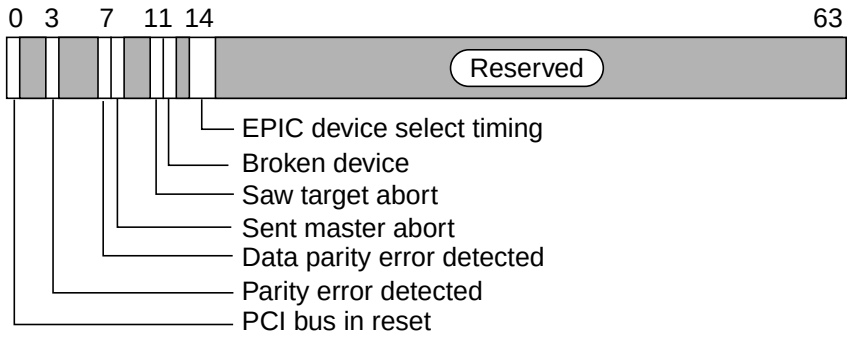
- *Arbitration disable timeout* field (bits 0:15)—Defines the PCI arbitration disable timeout threshold.
- *Reset PCI bus* field (bit 23)—Resets the PCI bus.
- *PCI memory I/O boundary* field (bits 29:31)—Controls the amount of I/O space mapped to PCI MEM and PCI I/O address space in contiguous 64-Mbyte blocks.
- *PCI memory space limit* field (bits 38:47)—Resets to 0x3f0 or 1008 decimal. The EPIC reserves the upper 60-Mbytes (channels 1008-1022 inclusive) for PCI controller shared memory address space and the highest four Mbytes (channel 1023) for EPIC context SRAM. The EPIC does not respond to PCI Controller DMA from channels 1,022 down to the value stored in this register. This register value updates on write to the PCI Memory_I/O Boundary field, and is read-only.
- *Disable PCI retry counter* bit (bit 50)—Specifies the PCI retry counter is enabled.
- *Enable data byte swap during host access to PCI* bit (bit 51)—Causes host accesses to the PCI to swap data bytes.
- *Disable Host highest priority* bit (bit 54)—Indicates the host is participating in rotating priority with other devices.
- *Disable PCI bus arbiter timeout on bus grants* bit (bit 55)—Indicates that the PCI bus arbiter does not timeout on bus grants.

- *Master Parity Error Response* bit (bit 59)—Indicates that the EPIC performs its normal operation when it detects a parity error as bus master.
- *Generate bad address parity on PCI master writes* bit (bit 62)—Forces bad address parity out to PCI (for diagnostic use only).
- *Generate bad data parity on PCI master writes* bit (bit 63)—Forces bad data parity out to PCI (for diagnostic use only).

PCI Master Status register

Each EPIC has one PCI Master Status register that provides status information for the PCI Master interface. All fields are cleared by a reset except the EPIC Device Select Timing field; it is hardwired to the value one.

Figure 55 PCI Master Status register definition



The bits in the EPIC PCI Master Status register are defined as follows:

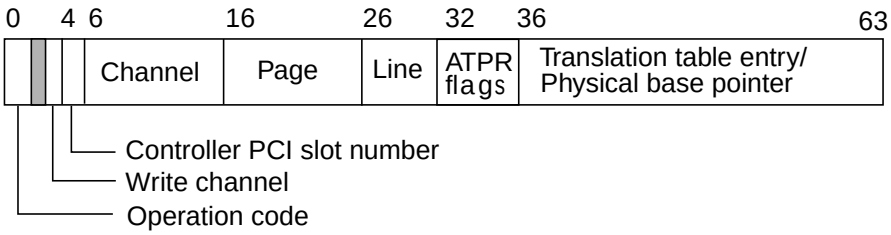
- *PCI bus in reset* bit (bit 0)—Indicates that the PCI bus is reset.
- *Parity error detected* bit (bit 3)—Indicates that the EPIC detected a parity error on incoming read data while the EPIC was bus master.
- *Data parity error detected* bit (bit 7)—Indicates that a parity error was detected on the bus while EPIC was PCI bus master.

- *Sent master abort* bit (bit 8)—Indicates that the EPIC was master and sent a Master Abort (no target claimed the bus cycle). The Host receives an error response.
- *Saw target abort* bit (bit 11)—Indicates that the EPIC was master and received a Target Abort. The Host will receive an error response.
- *Broken device* bit (bit 12)—Indicates that the EPIC PCI interface received a grant and an Idle bus for 16 clocks but did not run a bus cycle. The EPIC returns an error response to the requestor and sets the Master error in the Error Cause register.
- *EPIC device select timing* field (bits 14:15)—Sets medium-speed address decode on PCI. This field is read-only.

EPIC Channel Builder register

The EPIC Channel Builder register on each EPIC sets up channel context in preparation for an I/O operation. The format of the register is shown in Figure 56. Writing to this register stores the value to all fields, and reading it returns the last written value. A reset clears all fields. A read from the Channel Builder register returns the current state. A write to the Channel Builder register causes channel state to be modified as defined by the written data.

Figure 56 EPIC Channel Builder register definition



The fields and bits in the EPIC Channel Builder register are defined as follows:

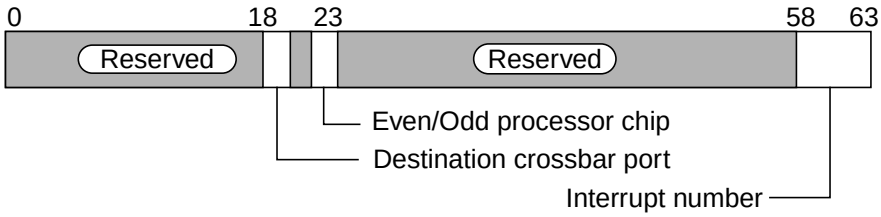
- *Operation code* field (bits 0:1)—Determines the operation the channel builder will perform.
- *Write channel* field (bit 3)—Indicates a memory write channel when set or a memory read channel when cleared.
- *Controller PCI slot number* field (bits 4:5)—Determines the PCI slot that this channel uses.

- *Channel number* field (bits 6:15)—Indicates the PCI channel to be built. The valid range is from 0 to the PCI memory space limit.
- *Page number* field (bits 16:25)—Indicates the 10-bit PCI page number.
- *Line number* field (bits 26:31)—Indicates the line number inside the page from which to start prefetch (only applies if a read channel, i.e. the Write Channel bit = 0).
- *A - Address translation enable* bit (bit 32)—Indicates the channel is in logical mode (translation on), if the A bit is set to a one value. If the A bit is set to a zero value, the channel is in physical mode (address translation off). This field also interprets the translation table base Pointer field.
- *T -TLB fetch enable* bit (bit 33)—If set, TLBs are fetched from memory; if T=0, only previously encached TLBs are available.
- *P -Prefetch/Write purge partial enable* bit (bit 34)—Indicates that data prefetch starts at the same time as channel build for read channels. It indicates that write purge partials are enabled for a write channel.
- *R -Refetch* bit (bit 35)—Enables data refetch prior to a read channel being swapped out.
- *Translation table entry/Physical base pointer* field (bits 36:63)—Indicates EPIC function as follows:
  - If the A bit is set to a one value and operation code is either a `Build` or `Init`, then the field is the translation table base pointer. This 28-bit field points to the translation table base address where TLBs are fetched.
  - If the A bit is set to a one value and operation code is a `Prefetch`, this field is the 28-bit TLE for the data prefetch.
  - If the A bit is set to a zero value the field is an 18-bit physical base pointer for this channel number, pointing to a four-Mbyte physically contiguous block of memory.

EPIC Interrupt Configuration register

The EPIC has one EPIC Interrupt Configuration register that specifies the interrupt number and processor when an interrupt occurs. The EPIC forwards the interrupt by writing the interrupt number to a local processor EIRR register. Since the EPIC can only send interrupts to one of the 16 processor EIRR registers on the system, only four bits of the address are programmable. All programmable fields are reset to zero.

Figure 57 EPIC Interrupt Configuration register definition



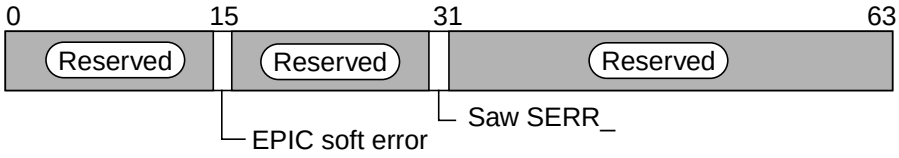
The fields and bits in the EPIC Interrupt Configuration register are defined as follows:

- *Destination crossbar port* field (bits 18:20)—Indicates the crossbar port (and therefore which EPAC) to which the interrupt will be sent.
- *Even/Odd processor chip* field (bits 23)—Specifies which of the two processors for the given EPAC the interrupt is to be sent.
- *Interrupt number* (bits 58:63)—Indicates the processor External Interrupt register interrupt bit to be set.

EPIC Interrupt Source register

Each EPIC has one Interrupt Source register that holds pending EPIC interrupts. Source bits are set when the source of the interrupt occurs and remains set until cleared. Interrupts are accumulated regardless of the state of the enable. If the interrupt is enabled in the EPIC Interrupt Enable register, then EPIC sends an interrupt. If the interrupt enable is written to a one value while the interrupt is pending in the Interrupt Source register, the EPIC generates an interrupt following the response to the register write.

Figure 58 EPIC Interrupt Source register definition



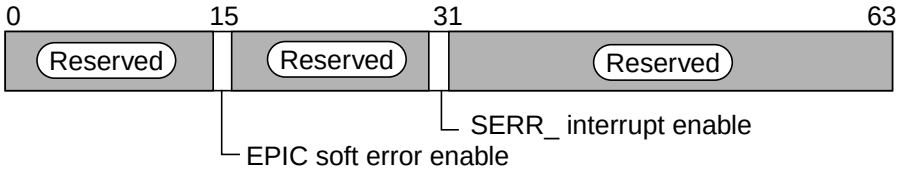
The bits in the EPIC Interrupt Source register are defines as follows:

- *EPIC soft error* bit (bit 15)—Indicates that a bit has been set in the Error Cause register that is configured as a soft error.
- *Saw SERR_* field (bit 31)—Indicates the EPIC received an SERR_ on the PCI bus.

EPIC Interrupt Enable register

EPIC Interrupt Enable register has a bit for every source interrupt in EPIC Interrupt Source. A one value in any EPIC Interrupt Enable bit causes an interrupt when the corresponding source event in EPIC Interrupt Source occurs. This register resets to zero (all interrupts disabled). The format for this register is shown in Figure 59.

Figure 59 EPIC Interrupt Enable register definition



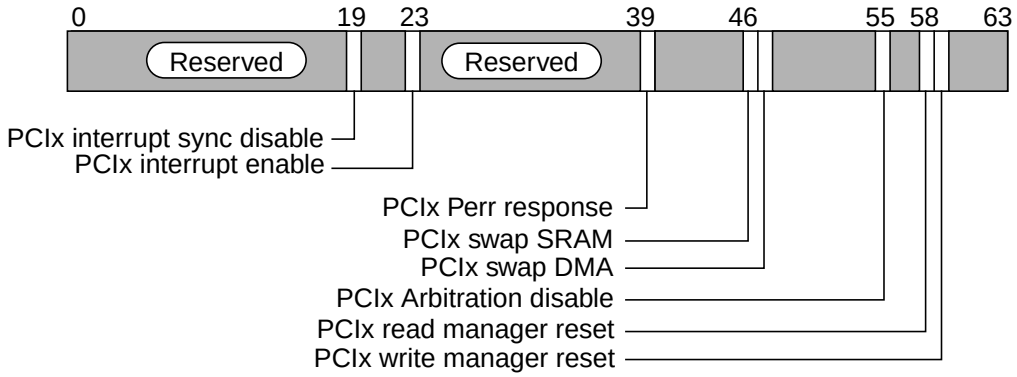
The bits in the EPIC Interrupt Enable register are defined as follows:

- *EPIC soft error enable* field (bit 15)—Indicates that an interrupt can be sent when a soft error occurs. A value of one enables the interrupt.
- *SERR_ Interrupt Enable* field (bit 31)—Indicates that an interrupt can be sent when a PCI SERR_ occurs.

PCI Slot Configuration register

There are four PCI Slot Configuration registers, one for each supported PCI expansion slot. These registers provide control of the PCI interface.

Figure 60 PCI Slot Configuration register definition



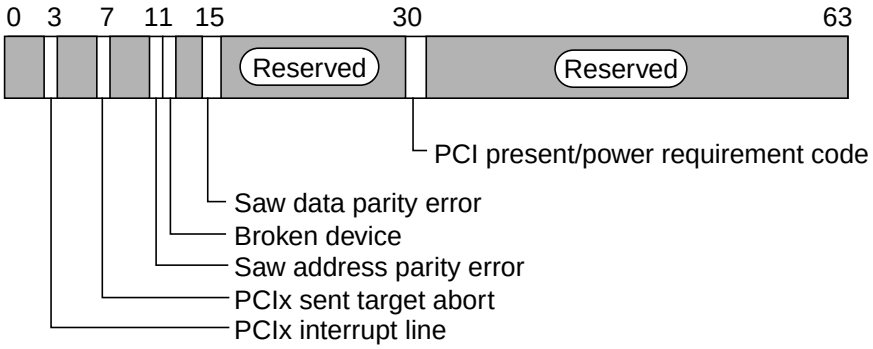
The bits in the PCI Slot Configuration register are defined as follows:

- *PCIx interrupt synchronization disable* bit (bit 19)—Disables the synchronization of a device on interrupt.
- *PCIx interrupt enable* bit (bit 23)—Enables the forwarding of a device interrupt.
- *PCIx Perr response* bit (bit 39)—Enables PCI_PERR_ data parity error signalling.
- *PCIx_Swap SRAM* bit (bit 46)—Enables byte swapping on PCIx shared memory transfers.
- P*CIx_Swap DM*A bit (bit 47)—Enables byte swapping on PCIx DMA transfers.
- *PCIx Arb Disable* bit (bit 55)—Disables bus arbitration for PCIx.
- *PCIx Read Manager Reset* bit (bit 58)—Resets EPIC Read manager x.
- *PCIx Write Manager Reset* bit (bit 59)—Resets EPIC Write manager x.

PCI Slot Status register

Each EPIC has four PCI Slot Status registers that specify the status of slot specific events. Bits in these registers are set when EPIC is the target of one of the four controllers and a status event occurs. All writable fields are reset to the value zero.

Figure 61 PCI Slot Status register definition



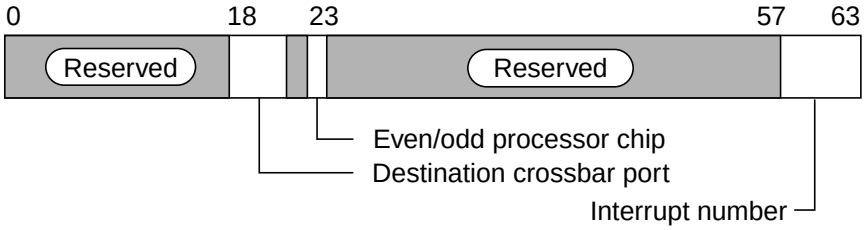
The fields and bits in the PCI Slot Status register are defined as follows:

- *PCIx interrupt line* bit (bit 3)—Indicates the current state of the PCIx INTA_ line (read only).
- *PCIx sent target abort* bit (bit 7)—Indicates that the EPIC sent this slot a Target Abort bus cycle termination. This bit does not set the PCI Controller x bit in the Error Cause register.
- *Saw address parity error* bit (bit 11)—Indicates that the EPIC detected an address phase parity error on a transfer from this slot. The EPIC terminates the transfer with target abort and relies on the transaction master to report the error to software. This bit sets the PCI Controller *x* bit in the Error Cause register.
- *Broken device* bit (bit 12)—Indicates this slot received a grant during an idle bus for 16 clocks but did not run a bus cycle.
- *Saw data parity error* bit (bit 15)—Indicates the EPIC (as a target) detected a PCI data phase parity error on incoming (write) data from this slot. This bit does *not* set the PCI Controller *x* bit in the Error Cause register.
- *PCI card present/power requirements code* field (bits 30:31)—Indicates a PCI controller is present and the power requirements of that controller. This field is read only.

PCI Slot Interrupt Configuration register

Each EPIC has four PCI Slot Interrupt Configuration registers that specify the interrupt number and processor when an interrupt occurs on the corresponding PCI slot. The EPIC forwards the interrupt by writing the interrupt number to a local processor EIRR register. Because the EPIC can only send interrupts to one of the 16 processor EIRR registers on the system, only four bits of the address are programmable. All programmable fields are reset to zero.

Figure 62 PCI Slot Interrupt Configuration register definition



The fields in the EPIC PCI Slot Interrupt Configuration register are defined as follows:

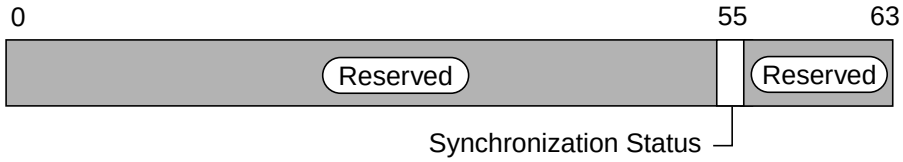
- *Destination crossbar port* field (bits 18:20)—Determines to which crossbar port (and therefore which EPAC) the interrupt will be sent.
- *Even/Odd processor chip* field (bits 23)—Specifies which of the two processors for the given EPAC the interrupt is to be sent.
- *Interrupt number* field (bits 58:63)—Specifies the processor External Interrupt register interrupt bit to be set.

PCI Slot Synchronization register

The EPIC has four PCI Slot Synchronization registers, one for each of the four PCI bus slots. Software polls these registers to determine when the write pipe has been flushed.

A processor reads these registers to synchronize the write pipe for the corresponding device. The registers are read-only, and the CSR interface returns zero status after the requested device operation completes. The format of the PCI Slot Synchronization register is shown in Figure 63.

Figure 63 PCI Slot Synchronization register definition



The *Synchronization status* bit (bit 55) specifies the completion status of the device write manager for the corresponding slot.

Byte swapping

In order to address different byte ordering between the PCI bus and the rest of the system, the EPIC provides CSR-configurable bits to define how to handle byte ordering of data crossing from one domain into the other. The CSRs configure byte swapping on the following data paths:

- PCI read and write of system coherent memory on a per-controller basis via the PCI Slot Configuration register (see the section “PCI Slot Configuration register” on page 128)
- PCI read and write of shared memory on a per-controller basis via the PCI Slot Configuration register
- Host read and write of PCI I/O, Memory, and Configuration space via the PCI Master Configuration register (see the section “PCI Master Configuration register” on page 121)

8 Performance monitors

This chapter discusses the hardware used to determine the performance of the system. Some performance factors include:

- Parallel program efficiency
- Communications costs
- I/O bandwidth
- Cache-hit rate

Performance factors

The performance of applications run on the V-Class server depends upon the factors already stated. The V-Class server includes hardware to measure each of the principal factors. The measurements indicate the overall results and provide data that enables programmers to identify changes to algorithms that improve overall performance. The process of measuring performance is intrusive and can impact system performance.

Some of the important factors and concepts for their measurement are:

- Efficient parallel algorithms—Parallel algorithms pose both validity and performance problems for the programmer and often prove more difficult to debug than single-threaded applications. Useful tools include:
  - Trace data correlated between threads
  - Deadlock detection
  - Synchronization statistics
  - Measurement of effective parallelism
  - Granularity measurements of parallel regions
  - Lock order enforcement
- I/O performance—The largest I/O factor is data transfer rates in the disk subsystem. Of particular interest are peak measurements, as most I/O is done in a burst mode. Also, information about disk access patterns should be available.
- Communication costs—Concerns in communication include the following:
  - Memory usage
  - Memory access patterns of particular code sections
  - Communication costs between threads
- Cache-hit rate—Algorithms must optimize cache use to perform well. Consequently, the V-Class server provides data on overall hit ratios, hit ratios per processor, hit ratios over time, cache miss trace data, and data that tells which program statements are causing the most misses.

Performance monitor hardware

The V-Class server provides registers to record events and enable performance measurement. These registers include:

- The Processor interval timer
- The time-of-century clock (TIME_TOC)
- The performance monitor set for each processor

Interval timer

Each processor has a 32-bit timer in control register 16. These timers count at a frequency between twice the peak instruction rate and half the peak instruction rate. They are not synchronized, nor can they be loaded by software. The timers may optionally generate an interrupt when the count reaches a certain value.

These timers are used for:

- Generating periodic clock interrupts to the processors for scheduling purposes.
- Measuring fine granularity time intervals within processors independent of other processors.
- Implementing thread timer register (TTR) in software. The operating system allows user-read access to this timer in order to use the TTR.

Time-of-Century clock

The V-Class server Time-of-Century clock (TIME_TOC) can be used to time-stamp trace data stored within the system. It also provides time-stamping of transmitted messages. The receiving processor can determine the transmission time by subtracting time-stamp from the current time.

Each EPAC has a 64-bit TIME_TOC register accessed with a single 64-bit read. The ECUB core logic generates a 16-Mhz clock for the each TIME_TOC register. The EPAC synchronizes the 16-Mhz clock to its own clock and generates a TIME_TOC clock every seven or eight EPAC clocks. The TIME_TOC logic generates a synchronization pulse every 256 TIME_TOC clocks.

A single EPAC acts as the TIME_TOC synchronization master of the system (the outputs of all other EPACs are in a high-impedance state). The master EPAC sends its synchronization pulse to the nonmaster EPACs.

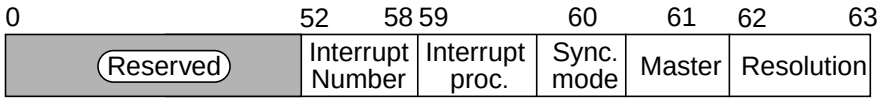
Logic ensures that the TIME_TOC registers maintain synchronization within their specified resolution. It checks to ensure the time between TIME_TOC synchronization pulses is in the range of TIME_TOC synchronization period plus or minus one-half the TIME_TOC synchronization resolution. If it detects a TIME_TOC synchronization pulse that occurs early or late, it sends an interrupt to one of the processors connected to the EPAC.

When the resolution field of the EPAC TIME_TOC configuration register has the value zero (TIME_TOC off), TIME_TOC synchronization pulse checking is disabled.

EPAC TIME_TOC Configuration register

The format of the EPAC TIME_TOC configuration register is shown in Figure 64.

Figure 64 EPAC TIME_TOC Configuration register definition



The bits and fields of the EPAC TIME_TOC Configuration register are defined as follows:

- *Interrupt number* field (bits 53:58)—Specifies the interrupt number sent to one of the two processors when a synchronization pulse check problem is detected. The field is not initialized by reset.
- *Interrupt processor* bit (bit 59)—Specifies the processor to which an interrupt is sent when a synchronization pulse check problem is detected. The field is not initialized by reset.
- *Synchronization mode* bit (bit 60)—Indicates whether the synchronization pulse starts incrementing or synchronizing the TIME_TOC and prescale registers. At reset the field is cleared and indicates that the next synchronization pulse received will start incrementing the TIME_TOC and prescale registers. The reception of

a synchronization pulse sets this bit. When the synchronization mode field is set, the reception of a synchronization pulse causes the prescale and least significant bits of the TIME_TOC register to be rounded.

- *Master* bit (61)—Specifies that the EPAC is the TIME_TOC master. The EPAC generates and delivers the synchronization pulse to the other EPACs. A value of one enables this operation. The field is reset to the value zero.
- *Resolution* field (bits 62:63)—Specifies the TIME_TOC register resolution. The field is reset to the value zero (TIME_TOC off). Table 22 shows the supported resolutions.

Table 22 TIME_TOC resolutions

Value	Resolution
0	TIME_TOC off
1	1 microsecond
2	2 microseconds
3	4 microseconds

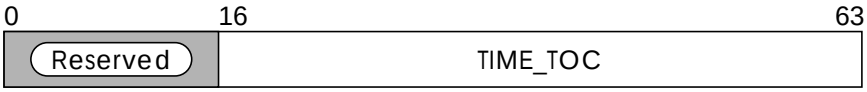
A value of zero (TIME_TOC off) disables TIME_TOC synchronization pulse checking.

EPAC TIME_TOC Clock register

The TIME_TOC Clock register is a 64-bit, read-write register of which only the least significant 48 bits are implemented (48 bits supports an uptime of 8.9 years). Read access supports normal operation, and write access supports initialization as well as testing.

The TIME_TOC register increments its 48-bit value when the pre-scale register reaches the value 0xF. The format of the EPAC TIME_TOC register is shown in Figure 65.

Figure 65 **TIME_TOC Clock register definition**



The *TIME_TOC* field (bits 16:63) increments each time the prescale logic has the value of 0xF. The register is accessible using a 64-bit CSR read or write. Reset does not effect this register. The least significant two bits may be rounded up or down when a synchronization pulse is received, depending on the resolution selected for the TIME_TOC logic.

TIME_TOC reset and initialization

Reset has the following effect on the TIME_TOC logic:

- The synchronization mode bit of the EPAC TIME_TOC configuration CSR register is cleared, forcing the prescale register to the value zero.
- The TIME_TOC register is inhibited.
- The master field of the EPAC TIME_TOC configuration register is cleared. With this field set to zero, the TIME_TOC synchronization pulse is disabled.
- The resolution field of the EPAC TIME_TOC configuration CSR is cleared, disabling the TIME_TOC synchronization checking logic.

Performance monitoring counters

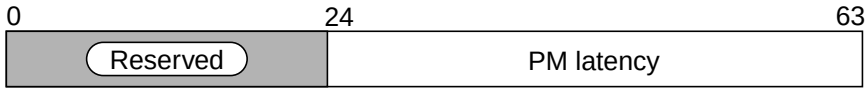
The PA-8200 processor has extensive performance monitoring capabilities on board. It does require, however, minimal additional hardware.

Each EPAC has two performance monitoring counters for both processors that includes a latency counter and a memory access event counters.

Latency counter

The latency counter is a 40-bit read-write register that increments by the number of outstanding processor data reads (0 to 10) at the system clock frequency. The EPAC has two latency registers, one for each processor. Figure 66 shows the bit definition of the Performance Monitor Latency Pn register counter (where n is the processor number).

Figure 66 EPAC Performance Monitor Latency register definition

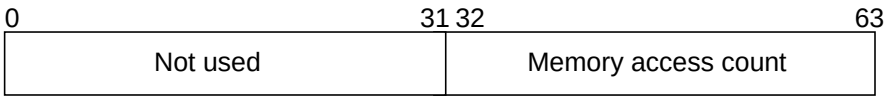


The *PM latency* field (bits 24:63) accumulates the total latency of all coherent read requests.

Event counters

The EPAC has a 32-bit read-write memory access event counter shown in Figure 67.

Figure 67 EPAC Performance Monitor Memory Access Count Pn register definition



9 System utilities

Each V-Class server has a section of hardware known as the Exemplar Core Utilities board (ECUB) located on the ENRB. On the ECUB are two FPGAs: the Exemplar Processor Utilities (EPUC) and the Exemplar Monitoring Utilities (EMUC). The EPUC provides the ECUB a means to send interrupts and error messages to the processors and to receive control messages from the processors. The EMUC performs all environmental monitoring. The ECUB board connects to all EPACs through the core logic bus.

Utilities board

The ECUB, or Utilities board, handles all system housekeeping chores. It connects directly to the ENRB where it attaches to the core logic bus, the environmental sensors, and other test points. It interfaces to the liquid crystal display (LCD), the optional teststation (an ethernet connection), and other external devices. Figure 68 shows the Utilities board functional layout.

The heart of the ECUB is the core logic. This section of hardware connects internally with the EMUC for receiving environmental interrupts and to the EPUC as an interface to the core logic bus. The core logic contains initialization and booting firmware. It also interfaces to the LCD and to serial RS232 links, as well as to ethernet links. An optional teststation can be connected via these links to run diagnostics and configure the system.

The EMUC latches system interrupts, most of which are from environmental sensors located throughout the system. The EMUC and the power-on circuit together control system power-up. The EMUC interfaces to a light-emitting diode (LED) diagnostic display through the power-on circuit.

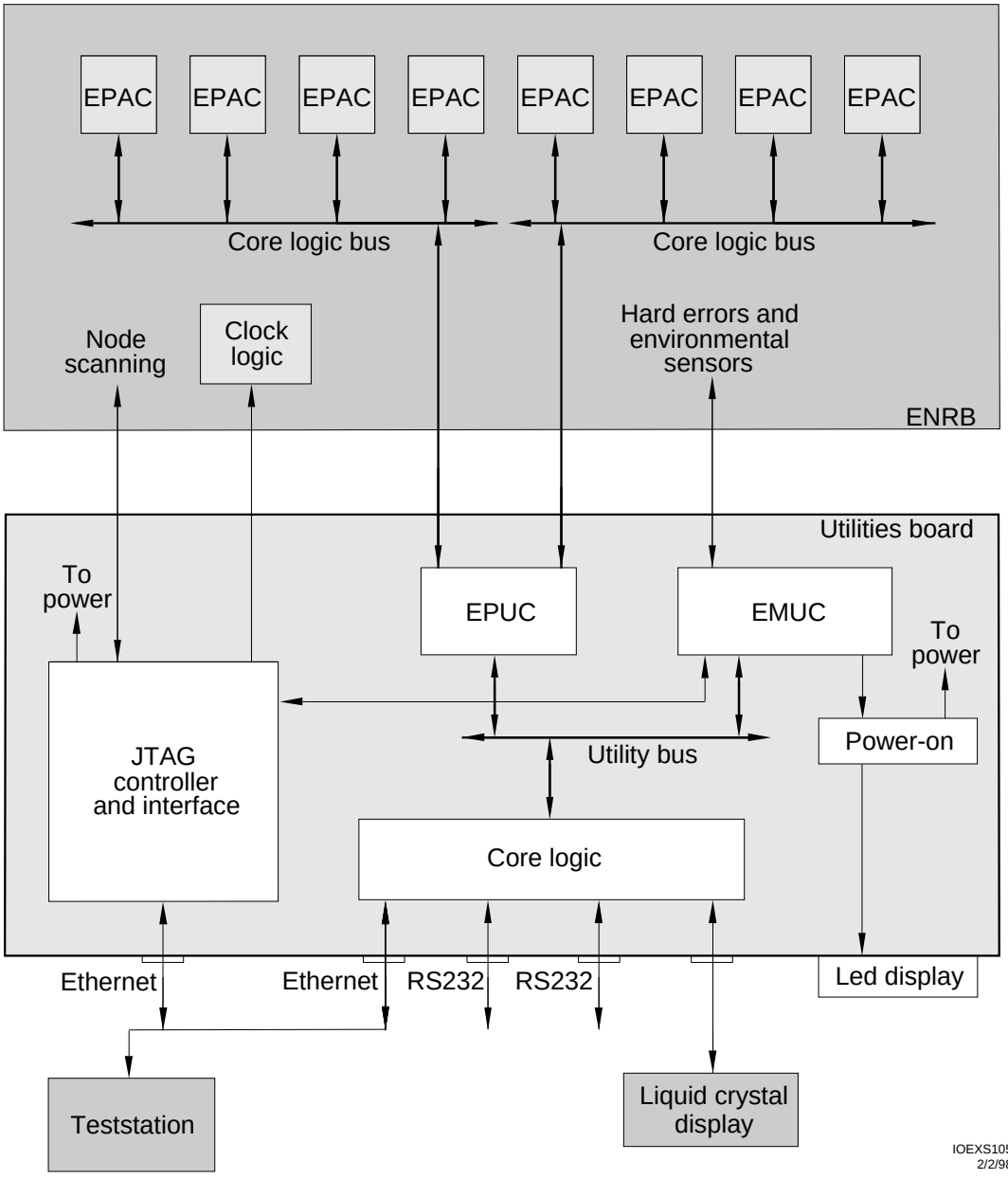
The EPUC provides the core logic an interface to the core logic bus. There are actually two buses; each one connects up to four EPACs. The EPUC communicates to the EPACs using data packets.

The JTAG (Joint Test Action Group) interface supports a teststation and a mechanism to fan out JTAG to all the boards in the system. It is used only for testing.

The V-Class server uses a test method called scanning to test boards and other hardware units. With the teststation connected to the ethernet between nodes, you can test any part of the system.

The JTAG interface contains a microprocessor to capture packets from the ethernet and apply them to the JTAG test bus controller or to take scan information from the JTAG test bus controller and send it out on the ethernet. The teststation can also read and write every CSR in the system.

Figure 68 Utilities board



Core logic

This section describes the core logic bus and core logic hardware functions.

Flash memory

The core logic contains nonvolatile storage for processor-dependent code. This code consists of primary loader code, the Open Boot PROM (OBP) code, the OBP interface firmware, spp_pdc, and power-on self test software (POST) (see the chapter "Booting and testing," for more information). This EEPROM memory is four MBytes, configured as one-million addresses by 32 data bits with only 32-bit read and write accesses allowed. It is writable by the processors for field upgrades and can be written when the EPUC is *scanned*.

Nonvolatile static RAM

The core logic section contains a nonvolatile battery-backed static RAM (NVSRAM). The NVSRAM is used to write system log information (failures) and store configuration information. This RAM is byte addressable and can be accessed even after power failures occur.

DUART

The ECUB logic contains a Dual Universal Asynchronous Receiver-Transmitter (DUART). One port, configured as a basic RS232 port, provides an interface to the simplest core system functions. With this interface, you can connect a terminal as a local console to analyze problems, reconfigure the system, or provide other user access. The parallel port of the DUART drives the LCD. The second RS232 port can be connected to a modem for field service.

RAM

RAM is needed to support the simple core system functions. When the system powers up, the processors operate out of this RAM. They run self test software to test and configure the rest of the system. Once the

system is fully configured, the processors execute out of main memory. The RAM is byte addressable and is 128 KBytes, configured as 32K addresses by 32 data bits (with parity).

Console ethernet

The ethernet I/O port connects to another optional system console that has an ethernet port. You can use the console for initializing, testing, and troubleshooting the system.

LEDs and LCD

LEDs display environmental information, such as the source of an environmental error that caused the ECUB to power down the system.

The LCD is driven by one of the processors via the ECUB. A large amount of information can be displayed on the LCD. The core logic drives the LCD via the parallel port on the DUART.

COP interface

COP chips (serial EEPROMs) are located on the major boards with information such as serial number, error history, configuration information, and so on. The EMUC connects to the COP bus selector (CBS) chip on the ENRB and allows the system to read any COP in the system.

EPUC

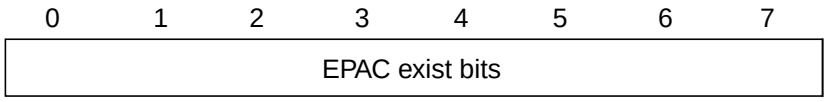
The EPUC applies interrupts and error messages to the processors and receives control messages from the processors. It has two 18-bit, bidirectional buses. Each interface connects up to four EPACs. The EPUC provides core logic bus arbitration for the sixteen processors.

Through the EPUC, the EPAC has an interface to the core logic bus on the ECUB. This bus connects the EPUC, the EMUC, and the core logic section together.

EPUC Processor Agent Exist register

The Processor Agent Exist register indicates which EPACs exist in the system. During reset, all EPACs assert their REQ lines. This sets corresponding bits in this register. The EPUC ignores the REQ lines (with respect to core logic bus requests) approximately eight clocks after reset to allow the EPACs to change from *exist* mode to *request* mode.

Figure 69 EPUC Processor Agent Exist register definition

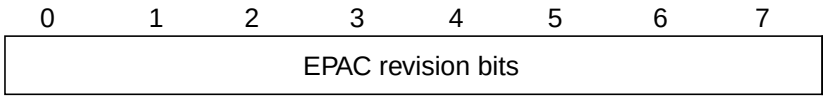


A value of one on any bit indicates that the respective EPAC 0-7 is active and exists.

EPUC Revision register

The Revision register indicates the revision level of the EPUC FPGA.

Figure 70 EPUC Revision register



Revision bits are read to determine the revision of the EPAC.

EMUC and Power-on

The EMUC performs all environmental monitoring on the ECUB. It attaches to the core logic bus so that processors can monitor the system by accessing these CSRs.

The EMUC works in conjunction with a hardware section on the ECUB known as the power-on circuit. This circuit controls powering up the entire system. It operates when the rest of the system is powered off or in some indeterminate state. It drives the environment LED display which is a basic (minimal hardware, no software) indication of what environmental error caused the ECUB to power down the system.

The teststation can also read the environmental LED display.

Environmental monitoring functions

The EMUC and the power-on circuit monitor the following environmental conditions:

- ASIC installation error sensing
- FPGA configuration and status
- Thermal sensing
- Fan Sensing
- Power failure sensing
- 48V failure
- 48V maintenance
- Ambient air temperature sensing
- Power-on

Table 23 **Environmental conditions monitored by the EMUC and power-on circuit**

Condition	Type	Action
ASIC Not Installed OK	Environmental error	Power not turned on, LED indication
FPGA not OK	Environmental error	Power not turned on, LED indication
48V fail	Environmental error	Power turned off, LED indication
Midplane power fail	Environmental error	Power turned off, LED indication
Board over temp	Environmental error	Power off in one second, LED indication, Interrupt
Fan not turning	Environmental error	Power off in one second, LED indication, Interrupt
Ambient air hot	Environmental error	Power off in one second, LED indication, interrupt
Other power fail	Environmental error	Power off in one second, LED indication, interrupt
Ambient air warm	Environmental warning	LED indication, interrupt
48V maintenance	Environmental warning	LED indication, interrupt
Hard error	Hard error	LED indication, interrupt

Environmental conditions detected by power-on function

The power-on function detects environmental errors (such as ASIC Install or FPGA Not OK) immediately and does not turn on power to the system until the conditions are corrected. It also detects environmental errors such as 48V Fail while the system is powering up and Midplane Power Fail after the system has powered up. If a failure is detected in these two cases, the power-on circuit turns off power to the system.

Environmental warnings such as 48V maintenance are also detected by the power-on circuit. It applies these to the EMUC, which then sends an environmental warning interrupt to the system processors.

In all cases, the power-on circuit lights an environmental LED display code. The environmental LED display code is prioritized so that it only displays the highest priority error or warning.

Environmental conditions detected by EMUC

The EMUC detects most of the environmental conditions. It samples error conditions during a time period derived from a local 10-Hz clock that drives the power-on circuit. It registers all the environmental error conditions twice and then ORs them together. If the conditions persist for 200 milliseconds, the environmental error bit is set, and an environmental error interrupt is sent to the EPUC, which sends it on to the processors. The EMUC then waits 1.2 seconds and commands the power-on circuit to power down the system.

This same procedure exists for an environmental warning except that an environmental warning interrupt is sent and the circuit does not power down the system.

The environmental error interrupt and the 1.2 second delay provide the system adequate time to read CSRs to determine the cause of the error, log the condition in NVRAM, and display the condition on the LCD.

After the system is powered down, the ECUB is still powered up, but all outputs are disconnected from the system.

Environmental LED display

Second-level registers in the EMUC drive the 6-bit display. The EMUC prioritizes the environmental errors and warnings and passes the information to the power-on circuit. This circuit prioritizes the 6-bit field with its environmental conditions and produces a 7-bit field plus an attention bit (ATTN) that drives the Display. ATTN is on if there is an environmental warning.

In general, the power-on-detected errors are a higher priority than EMUC-detected errors, the lower the error code number, the higher its priority. Environmental warnings are lower priority than the environmental errors. Table 24 shows the LED display error codes.

Table 24 **Environmental LED display**

ATTN bit	LED Display	Description
1	00	ECUB 3.3V error (highest priority)
1	01	ASIC Install 0 (ENRB)
1	02	ASIC Install 1 (MEM)
1	03	FPGA not OK
1	04-07	DC OK error (UL, UR, LL, LR)
1	08-11	48V error, NPSUL fail, PWRUP=0-9
1	12-1B	48V error, NPSUR failure, PWRUP=0-9
1	1C-25	48V error, NPSLL failure, PWRUP=0-9
1	26-2F	48V error, NPSLR failure, PWRUP=0-9
1	30-39	48V error, no supply failure, PWRUP=0-9
1	3A	48V 7yo-yo error
1	3B	ENRB power failure (ENRBPB)
1	3C	Clock failure
1	3D-3F	Not used (3)
1	40-47	MB0-MB7 power failure
1	48-4F	PB0L, PB1R, PB2L, PB3R, PB4L, PB5R, PB6L, PB7R power failure
1	50-57	PB0R, PB1L, PB2R, PB3L, PB4R, PB5L, PB6R, PB7L power failure (possibly switch R and L)
1	58-5B	IOB (LR,LF,RF,RR) power failure
1	5C-61	Fan failure (UR,UM,UL,LR,LM,LL)

ATTN bit	LED Display	Description
1	62	Ambient hot
1	63	Overtemp ENRB
1	64-67	Overtemp quadrant (RL, RU, LL, LU)
1	68	Hard error
1	69	Ambient warm
1	6A-6F	Not used (6)
1	70-73	DC supply maintenance (UL,UR,LL,LR)
1	74-7F	Not used (12)
0	00-09	PWRUP state (00=System all powered up), attention LED off

The top of the table is the highest priority, the bottom the lowest. If a higher condition occurs, that one is displayed.

Monitored environmental conditions

This section describes each environmental condition that is monitored by the power-on circuit and the EMUC.

ECUB 3.3V error

This error indicates that the ECUB 3.3V power supply has failed, but the 5V supply has not.

ASIC installation error

Each ASIC has *install* lines to prevent power-up if an ASIC is installed incorrectly (such as an EPAC installed in an ERACs position). If an ASIC is improperly installed, the ECUB does not power up the system. This condition is not monitored after power up.

DC OK error

When this error is displayed, the power-on circuit did not power up the system, because one or more 48V power supplies reported an error. In systems with redundant 48V power supplies, this error means that two or more 48V supplies reported an error.

48V error

If the 48V supply has dropped below 42 volts for any reason other than normally turning off the system or an ac failure, then this error is displayed by the power-on circuit. Also, the 48V supply that reported the error and the power-up state of the system at the time of the error is displayed.

48V yo-yo error

This error indicates that a 48V error occurred and the ECUB lost and then later regained power without the machine being turned off. The power-on circuit will display this error and not power on the system, because the 48V supply is likely at fault.

Clock failure

If the system clock fails, then the EMUC will be unable to monitor environmental errors that could possibly damage the system. If the power-on circuit receives no response from the EMUC, it powers down the system and displays this error.

FPGA configuration and status

The EMUC is programmed by a serial data transfer from EEPROM upon utility board power-up. If the transfer does not complete properly, the EMUC cannot configure itself and many environmental conditions cannot be monitored. The power-on circuit monitors both the EMUC and EPUC and does not power up the system, if they are not configured correctly.

Board over-temperature

There is one temperature sensor per board that detects board overheating. The sensors are bussed together into four system quadrants plus the ENRB and applied to the EMUC.

Fan sensing

Sensors in the six fans determine if the fans are running properly. The EMUC waits 12.8 seconds for the fans to spin up after power-up before monitoring them.

Power failure

Because a power failure on a board could cause damage to other boards, a mechanism is in place to detect 3.3V failures on each board. Power failures are considered environmental errors, and the system is powered down after they are detected.

ENRB power failure

If the ENRB power fails, the power-on circuit powers down the entire system. The ECUB is still active, but the power-on circuit displays the power failure condition and disables all ECUB outputs that drive the system. This condition persists until power is cycled on the ECUB.

48V maintenance

There are up to four 48V power supplies. Each sends a signal to the power-on circuit. If any supply fails at any time, the circuit asserts the 48V maintenance line to the EMUC, which reports the environmental warning to the processors. The power-on circuit displays the highest priority 48V supply that failed.

Ambient air sensors

The ambient air sensors detect a too warm or too hot condition in the input air stream. Ambient air too warm is an environmental warning; ambient air too hot is an environmental error that powers down the system.

The temperature set points are set by the teststation. The digital temperature sensor has nonvolatile storage for the temperature set points. Power-on reset starts the digital temperature sensor without the core logic microprocessor intervening.

Environmental control

Described in the following sections are functions the ECUB performs to control the system environment.

Power-on

When the power switch is turned on, the outputs of the 48V power supplies become active. Several hundred milliseconds after the ECUB 5V supply reaches an acceptable level, the power-on circuit starts powering up the other dc-to-dc converters of the system in succession.

The power-on circuit does not power up the system if an ASIC is installed incorrectly (see the section “ASIC installation error” on page 151) or if an FPGA is not configured (see the section “FPGA configuration and status” on page 152). It keeps the system powered up unless an environmental condition occurs that warrants a power-down.

Voltage margining

Voltage margin is divided into four groups to minimize control, but allows all boards that communicate with each other to be margined separately for nominal, upper, and lower voltage.

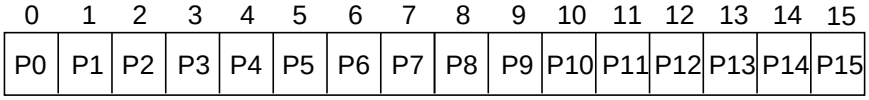
EMUC CSRs

This section describes some of the EMUC CSRs.

EMUC Processor Report register

The Processor Report register indicates the processors that are working in the system. Each processor reports by writing to this register and setting the bit corresponding to the processor number.

Figure 71 Processor Report register definition

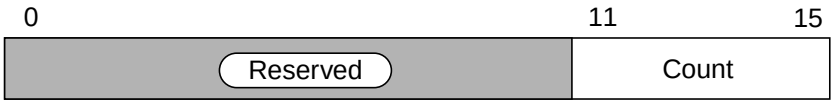


P0-P15 comprise a fully readable and writable field. The bits are cleared on reset. Once a bit is written to a one value, it remains set until cleared by reset. Writes of a zero value do nothing. The bit, P*x*, set to a one value, indicates that processor *x* has reported in working.

EMUC Processor Semaphore register

The Processor Semaphore register provides a signaling function for processor synchronization. This is an atomic read-and-increment register.

Figure 72 Processor Semaphore register definition

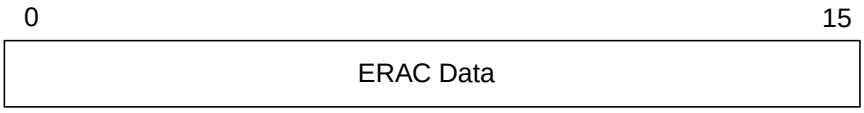


Count is cleared on reset. Writes load any value. Reads return the value of Count and then increment Count atomically.

EMUC ERAC Data register

The ERAC data register holds the data to be written to the destination ERAC CSR or the data that has been read from the ERAC CSR.

Figure 73 ERAC Data register definition

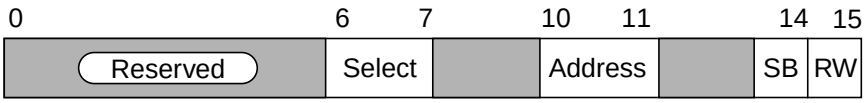


ERAC Data bits comprise a fully readable and writable field. After an ERAC read operation, the ERAC Data register holds the data. After the ERAC write operation, the data is stored in the ERAC register, and ERAC Data is undefined.

EMUC ERAC Configuration Control register

The ERAC Configuration Control register selects the target ERAC, the address of the CSR within that ERAC, and the type of CSR access (read or write). It controls the ERAC CSR operation and then returns status of the operation.

Figure 74 **ERAC Configuration Control register definition**



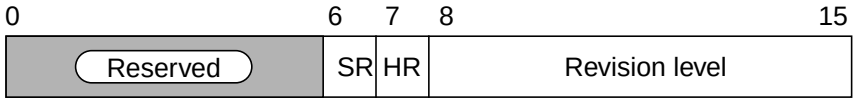
The fields and bits of the ERAC configuration Control registers are defined as follows:

- *Select* field (bits 6:7)—Selects the target ERAC. This field is write only.
- *Address* field (bits 10:11)—Selects the address of the CSR within that ERAC.
- *SB Start/Busy* bit (bit 14)—Starts the ERAC operation by writing a one value. Reading the bit returns the status of the operation (0=idle, 1=busy). SB and SEL must be written together.
- *RW* bit (bit 15)—Selects the type of operation: Read (RW=1), or Write (RW=0).

EMUC Reset register

The EMUC Reset register initiates a reset or displays the type of the last reset. This CSR also contains the revision status.

Figure 75 **EMUC Reset register definition**



The bits and field of the Reset register are defined as follows:

- *SR* (Soft Reset) bit (bit 6)—Initiates a soft reset.
- *HR* (Hard Reset) bit (bit 7)—Initiates a hard reset.

  The combination of SR and HR bits in the read mode indicate the resets shown in Table 25. The combination of SR and HR bits in the write mode indicate the resets given in Table 26.

Table 25 Reset register read codes

SR HR - Read	Last reset was
0 0	Power-on reset
0 1	Hard reset
1 0	Soft reset

Resets are initiated by writing to this register. Reset is asserted according to the codes in Table 26. The only difference between a hard and soft reset is the action taken by the software upon reading the codes.

Table 26 Reset register write codes

SR HR - Write	Action taken
X 1	Hard reset
1 0	Soft reset

- *Revision* field (bits 8:15) indicates the revision of the EMUC FPGA. This field is read only.

JTAG interface

The JTAG interface supports a teststation and a mechanism to fan out JTAG to all the boards in a system. It is used only for testing.

The JTAG functions are described in the following sections.

Teststation interface

The teststation can be a PA-RISC based workstation. The interface to the teststation is an ethernet AUI port for flexibility in connecting to many workstations.

AC test

An ac test is performed by a Test Bus Controller (TBC) scanning in data to all boards in the system and loading an ac test instruction into all ASICs on one board.

Once all boards have been almost loaded with the ac test instruction and paused, the TBC takes all boards out of pause mode simultaneously causing them all to exit update together and execute the ac test.

The ac test enables clocks inside the ASICs so that they test internal and external paths at the system clock rate. They all execute on the same system clock.

Clock margining

Parallel ports on the core logic microprocessor select the nominal, upper, or external clock that drives the system.

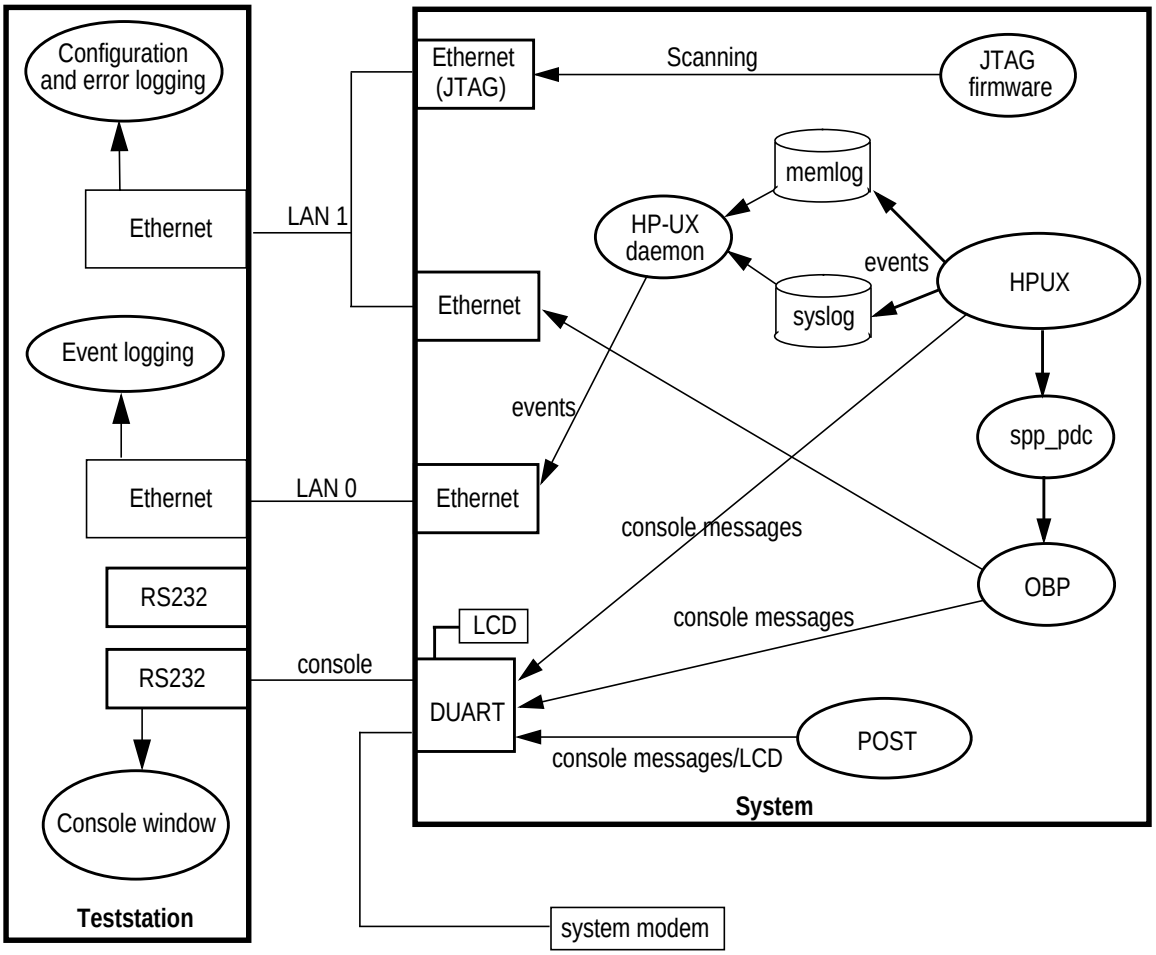
10 Booting and testing

The ECUB contains system booting and testing functions. It connects to the system teststation to provide a means of initialization, diagnostic testing, and remote booting of the system.

Teststation-to-system communications

This section describes how the teststation communicates with the system. Figure 76 shows the paths and processes for the V-Class server using HP-UX.

Figure 76 teststation-to-system communications



The hardware components located on the ECUB are shown in the diagram on the left side of the system. They include three ethernet ports and one DUART.

A layer of firmware between HP-UX and OBP called spp_pdc allows the HP-UX kernel to communicate with OBP. spp_pdc is platform-dependent code and runs on top of OBP providing access to the devices and OBP configuration properties.

LAN 0 communications

One system ethernet port connects to global LAN 0. Across this LAN system events are reported to the teststation using the HP-UX daemon. This daemon polls two databases, memlog and syslog, for errors and reports these errors on the teststation. HP-UX loads memory errors into the memlog database and recoverable cache errors into the syslog database.

LAN 1 communications

The two other ECUB ethernet ports connect to the system LAN 1. The JTAG port is used for scanning. The other port is used for downloading system firmware via nfs, downloading disk firmware, and loading firmware into the I/O subsystem. A configuration utility that is located on the teststation obtains system configuration information.

Serial communications

The DUART port on the ECUB provides an RS232 serial link to the teststation. Through this port HP-UX, OBP, firmware in EEPROM known as Power-On Self Test (POST—see the section “Power-On Self Test routine” on page 163) send console messages. POST and OBP also send system status to the LCD connected the DUART.

Booting

Booting a system refers to a sequence of events that loads and executes the operating system code. This sequence, or *boot procedure*, begins at power-on with the system in an unknown state and ends when the system begins executing the operating system.

Hardware reset

When power is applied to the system, all controllers receive a power-up reset signal. Hardware initialization occurs within the first few clocks after the reset pulse is negated.

The reset signal has the following effects:

- EPAC initialization—Hard error reporting is disabled, and all error registers hold their previous values if a hard error was logged before reset was applied. The identification number of each processor is loaded into a CSR. The registers can only be cleared by software.
- EPUC initialization—Hard error reporting is disabled, and all error registers are cleared. All other EPUC CSRs hold their previous values and can only be cleared by software.
- EPIC initialization—Hard error reporting is disabled, and all error registers hold their previous values. The registers can only be cleared by software.
- ERAC initialization—Hard error reporting is disabled, and all error registers hold their previous values. The registers can only be cleared by software. All ports are enabled.
- EMAC initialization—Hard error reporting is disabled, and all error registers hold their previous values if a hard error was logged before reset was applied. The registers can only be cleared by software.

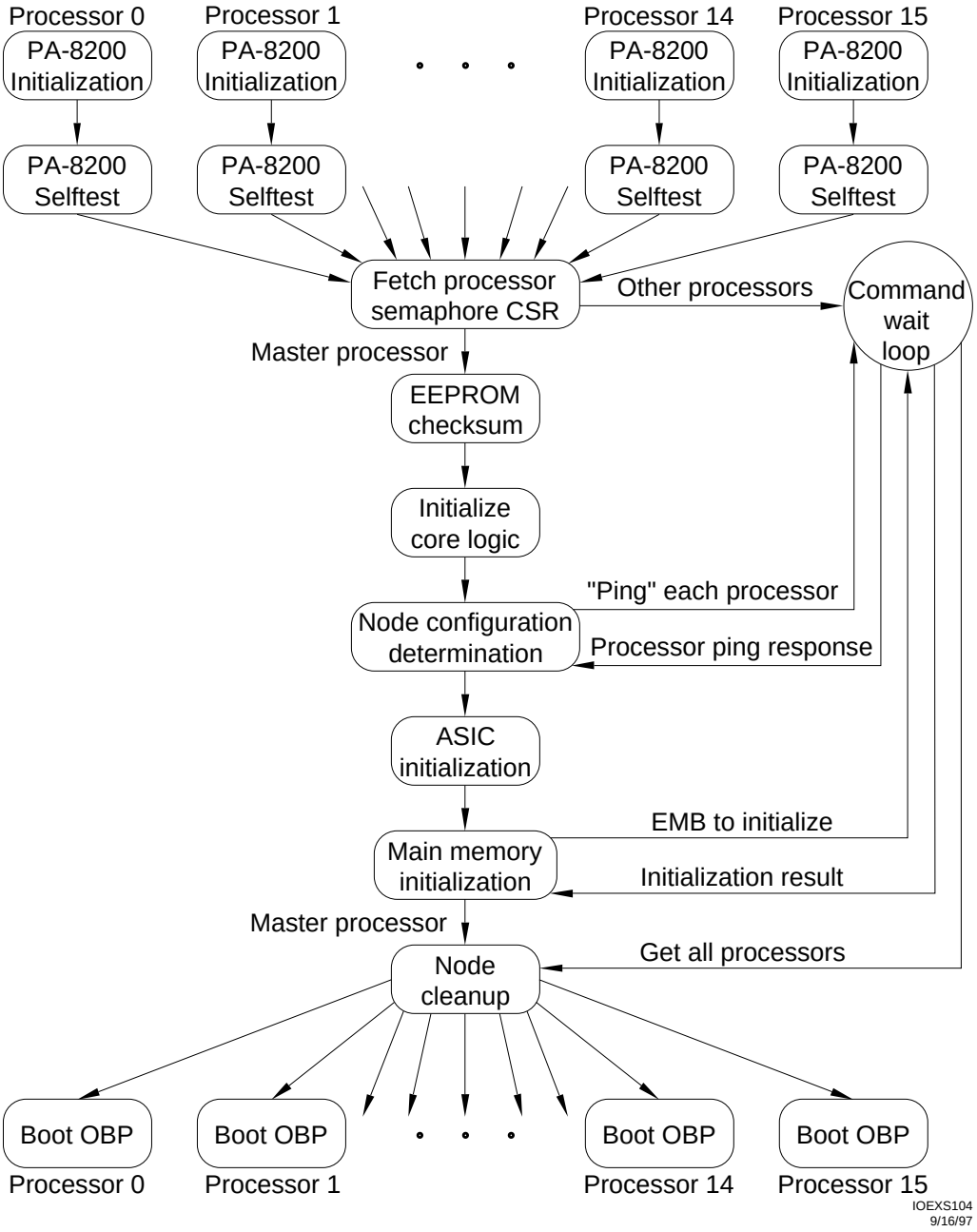
Power-On Self Test routine

When the system first powers up, all processors and supporting hardware must be initialized before the system proceeds with booting.

Upon power up, POST begins executing and brings up the system from an indeterminate state and then executes OBP. POST determines the system hardware configuration before running OBP. If POST encounters an error during initialization, it passes the appropriate error code to an LCD.

Figure 77 shows how POST initializes the processors up to booting of OBP.

Figure 77 POST program flow



Basic processor initialization and selftest

Upon reset, all processors are initialized and placed in selftest. The extent of the selftest is determined by a mode bit in NVRAM. If both the selftest and cache test NVRAM variables are enabled, cache testing is performed. Data cache initialization is verified, and the portion of the instruction cache used for memory initialization is pattern tested.

Each processor determines its identification (ID) from the EPAC. Also, each processor fetches the Processor Semaphore register on the PUC. Because register requests are queued, one processor will fetch this CSR before the others and becomes the booting, or *monarch*, processor. All others go into a command wait idle loop. The booting processor continues executing POST code from the EEPROM.

Checksum verification of the core logic NVRAM

The EPUC EEPROM contains the POST, OBP, system diagnostics code, and the spp_pdc code. In addition, these four routines use shared data structures. The routines and shared data structures all reside in sections of the EEPROM know as code spaces. Each code space has an embedded checksum word. POST checks the validity of each code space by reading and comparing its checksum.

Core logic initialization

The core logic contains SRAM and DUARTs that support external terminal connection for self test and the LCD panel. POST initializes the SRAM, DUARTs, and all controller CSRs in the system.

System configuration determination

POST determines which and how many controllers reside in the system (not every system contains a full complement of support hardware). It also determines the number of memory modules and their sizes. Any controller (ASIC) that does not respond to any CSR access is considered to be not installed.

System ASIC initialization

POST sets every system controller (ASIC) to a known state. The state is based both on configuration parameters and the current hardware configuration.

EPACs are reported in the EPUC EPAC-Exist register.

EMACs and EPICs are reported in the EPAC Configuration register.

ERACs are always all present (they are not sensed).

System main memory initialization

The processor reads the node ID from the COP EEPROM and uses its information to load the node identification register.

Next, the monarch processor determines the memory configuration for all EMACs. It determines the size, population, and installation of each DIMM on a memory board and returns this information to POST. The results are compared with the results of each other memory mapping and the least common denominator is determined and mapped in. Once the memory population is determined, the monarch processor assigns available processors to the enabled EMBs, initializing memory and tags in parallel.

System clean up and OBP boot process

POST resets the EPUC Processor Semaphore CSR and cleans up any residual state information from the initialization process. All processors now begin to execute the OBP routine at approximately the same time.

HP-UX bootup

Once each processor in the system has completed initialization and selftest, it loads and executes OBP. The following is the sequence of events for booting the system starting with loading OBP (for every processor) and finishing with the system ready for use:

- The processor loads OBP—After initialization and selftest, each processor loads and begins executing OBP. OBP transfers its ROM image to RAM, initializes the virtual mode, and turns on translation.
- OBP builds its device tree—It probes the system hardware.
- The processor loads spp_pdc from flash RAM—This firmware is layered over OBP and provides interface between OBP and the HP-UX kernel. spp_pdc must be loaded before OBP can perform any boot functions.
- OBP loads system boot loader—It opens the boot disk, loads a special system loading program, and closes boot disk.
- The processor executes spp_pdc—This firmware layer must be executing so that OBP can complete booting the system.
- The processor executes system boot loader—The loader starts in physical mode (32 bits) and performs the following tasks:
  - Relocates itself
  - Opens PCI devices through spp_pdc

    When spp_pdc calls OBP to perform PCI I/O transfers, OBP must turn on its virtual mode and then turn virtual mode off again when it returns control to spp_pdc. This means all buffers must already be equivalently mapped in OBP's virtual mode page tables.
  - Reads in the kernel using spp_pdc for I/O
  - Starts the kernel
- The kernel reads /etc/ioconfig—spp_pdc opens the boot device for I/O.
- The kernel boot I/O completes.
- spp_pdc closes the boot device.
- OBP turns off Virtual Mode—It removes PCI CSR virtual mode mapping.

- The kernel switches to its virtual mode
- The kernel relocates the system boot loader
- Kernel continues booting in one of two ways: normal boot and install boot.

Normal booting

For normal booting, the following additional tasks are performed:

- OBP loads the special system kernel loader into memory.
- The kernel loader loads /stand/vmunix or user-specified kernel.
- The kernel uses kernel loader for boot I/O to load /etc/ioconfig. Booting is complete.

Install booting

For install booting, the following sequence is performed:

- OBP loads the kernel loader into memory.
- The kernel loader loads `VINSTALL` LIF image.
- `VINSTALL` uses the kernel loader for boot I/O to load ramdisk `VINSTALLFS`
- `VINSTALL` completes booting, and the cold install GUI opens for the user.

Testing

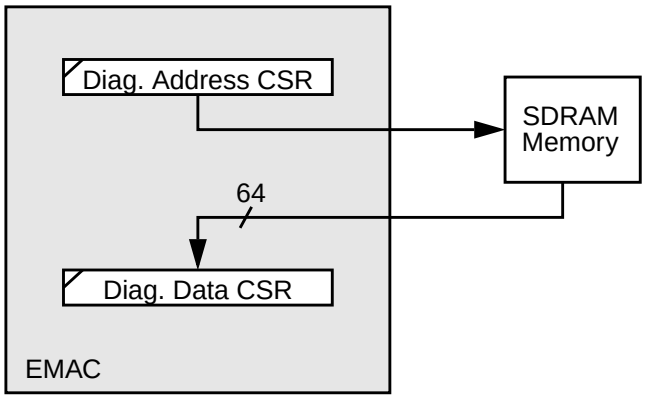
The system uses LAN1 for testing, running diagnostics on the system, and reconfiguring the system manually. All diagnostic accesses to memory occur through CSR space. A 64-bit register holds eight bytes for writing and reading memory. The memory transfer size is 64 bits. The diagnostic operations are:

- Memory line data read
- Memory line data write
- Memory line initialization
- Memory read ECC
- Memory write ECC
- Memory line scrub

Diagnostic memory read Operations

A diagnostic memory read is performed by writing to the memory line address of the Diagnostic Address CSR on the EMAC. The EMAC interprets the write to the CSR address as a request to read the data at the addressed memory line. The 64-bit data is read from the SDRAM memory and written to the EMAC diagnostic data register. The processor requesting the memory read can then access the data with a 64-bit CSR access. Figure 78 illustrates the flow for a CSR memory read operation.

Figure 78 **CSR memory read operation**



Diagnostic memory write operations

A diagnostic memory write operation uses the same CSRs as the read. The data is written to the CSR data register, and the address at which the data is to be stored is written to the CSR diagnostic address register.

EMAC diagnostic CSRs and addresses

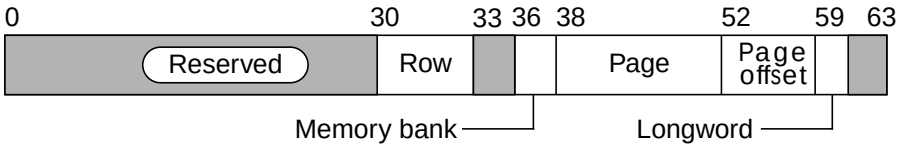
This section defines some of the EMAC CSRs that perform the diagnostic memory operations.

EMAC Diagnostic Address register

There is one Diagnostic Address register on each EMAC that supplies the address for diagnostic memory accesses.

The format of the Diagnostic Address register is shown in Figure 79.

Figure 79 **EMAC Diagnostic Address register definition**



The *Row* field (bits 30:32), *Memory bank* field (bits 36:37), *Page* field (bits 38:51), *Page offset* field (bits 52:58), and *Longword* field (bits 59:60) together specify the memory address for diagnostic accesses.

The *Longword* bit is used for diagnostic memory data reads and writes where a specific eight-byte longword must be accessed.

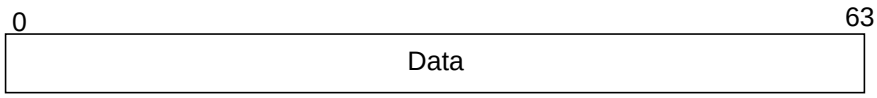
Bank interleaving is not performed by the EMAC; the processor must perform the mapping from the virtual bank to the memory bank.

All fields are written by a CSR write and read by a CSR read. A Diagnostic Memory Initialization operation increments the concatenated Page and Page Offset fields as part of the operation.

EMAC Diagnostic Data register

The Diagnostic Data register holds eight bytes of data used for diagnostic memory reads and writes. The format of the Diagnostic Data register is shown in Figure 80.

Figure 80 EMAC Diagnostic Data register definition



EMAC Diagnostic Read Memory Data address

Writing to the EMAC Diagnostic Read Memory Data address obtains eight bytes of data from the address specified by the Diagnostic Address register. The Diagnostic Address register is not modified by this operation.

EMAC Diagnostic Write Memory Data address

Writing to the EMAC Diagnostic Write Memory Data address moves eight bytes of data to the address specified by the Diagnostic Address register. The operation is similar to the read memory data operation, except that the Diagnostic Address register also specifies which of the four eight-byte longwords of a 32-byte line is to be written with the data in the Diagnostic Data register. Neither the Diagnostic Address nor Diagnostic Data register is modified by the operation. ECC is regenerated for the entire memory line by this operation.

EMAC Diagnostic Memory Read ECC address

The data in the memory line is stored in four consecutive SDRAM locations. Each SDRAM location is protected with an eight-bit ECC. Writing to the EMAC Diagnostic Memory Read address triggers a memory read operation of the ECC associated with the address specified in the Diagnostic Address register. The Longword field of the Diagnostic Address register specifies which ECC (of the four SDRAM locations) is to be read. The accessed ECC is written to the eight least significant bits of the Diagnostic Data register. The Diagnostic Address register is not modified by the operation.

EMAC Diagnostic Memory Write ECC address

Writing to the Diagnostic Memory Write ECC address triggers a memory write operation of the ECC associated with the address specified in the Diagnostic Address register. The Longword field of the Diagnostic Address register specifies which of the four SDRAM locations the ECC is to be written. The SDRAM ECC is written with the least significant eight bits of the Diagnostic Data register.

EMAC Diagnostic Memory Initialization address

Writing to the EMAC Diagnostic Memory Initialization address triggers a memory write operation to the tag and data of a memory line. The address of the memory line is specified by the Diagnostic Address register. The memory tag is written with the contents of the Diagnostic Data register, and the 32 bytes of memory data associated with the memory line are written with the value zero. The Diagnostic Data register is not modified. The concatenated Page and Page Offset fields of the Diagnostic Address register are incremented to address the next sequential memory line. The Longword field of the Diagnostic Address register is ignored.

EMAC Diagnostic Scrub Memory address

Writing to the EMAC Diagnostic Scrub Memory address triggers a read and write to the memory line at the address specified by the Diagnostic Address register. If a single bit ECC error occurs when the data is read, the data is corrected before it is written back into memory. If no ECC error occurs, the data is written back unmodified.

11 Error handling

An error (or fault) is an abnormal condition with hardware or firmware (processor-dependent code); the cause of the abnormality can be either transient or permanent. The cause can also be classified as a recoverable (soft or advisory) error or an unrecoverable (hard) error, depending on whether continued operation of the system is possible.

Most hardware faults are transient in nature, not being the result of a permanent hardware failure. The effect of these errors can many times be contained while the system continues to operate. There are, of course, occasions when hardware fails, continued operation is not possible, and the system must be taken down for diagnostic evaluation.

Soft errors

A recoverable error that results in the disabling of one or more processes but allows continued operation of the system is a soft error. An example of a soft error is a parity error on data read by a process. The process cannot continue, but the system continues to operate.

Soft errors can occur only during transactions that require a response. The error is reported to the requesting processor in one of two ways:

- The requesting processor detects the error itself (for example, a parity error).
- The detecting hardware sends an error response instead of its normal response. The error response contains some useful information about the error.

Whenever a soft error is reported to a processor, it invokes an HPMC. Any process running on a processor when an HPMC occurs is aborted by the operating system. If the process is a kernel or server, an operating system panic occurs. In this case, the system must be rebooted.

Advisory errors

A special type of recoverable error is an advisory error. Advisory errors are usually corrected by hardware or firmware. They are logged in the appropriate CSRs of the detecting controller and do not affect any processes running on the system. An example is a single-bit ECC memory error.

Advisory errors are not reported. Software must poll the CSRs periodically to determine their occurrence. Reading the CSRs when a soft error is detected can determine if it propagated from an earlier detected advisory error.

If a detected error causes corrupted data and another soft error is detected before corrupted data is consumed, it is also considered an advisory error. An example is a data parity error detected during a “responseless” request, such as a write-back, where corrupted data is written to memory. This error is logged as an advisory error. Any further reference to the line will cause a soft error, an indication that the data is corrupt.

Hard errors

Hardware can fail in such a manner that a process can receive corrupt data without detecting it. If an error is detected that prevents returning an error response or corrupts data so that future references cannot detect the corruption, it is considered a hard error. An example is a parity error detected on the address of a memory transaction. The appropriate memory line cannot be updated, and future consumers of the line can not be notified of the corruption.

A hard error is sent to the EMUC and then the EPUC may generate an interrupt, HPMC, or transfer of control (TOC) to one or all processors in the system. See “EPUC interrupt logic” on page 98 for more information on EPUC interrupts.

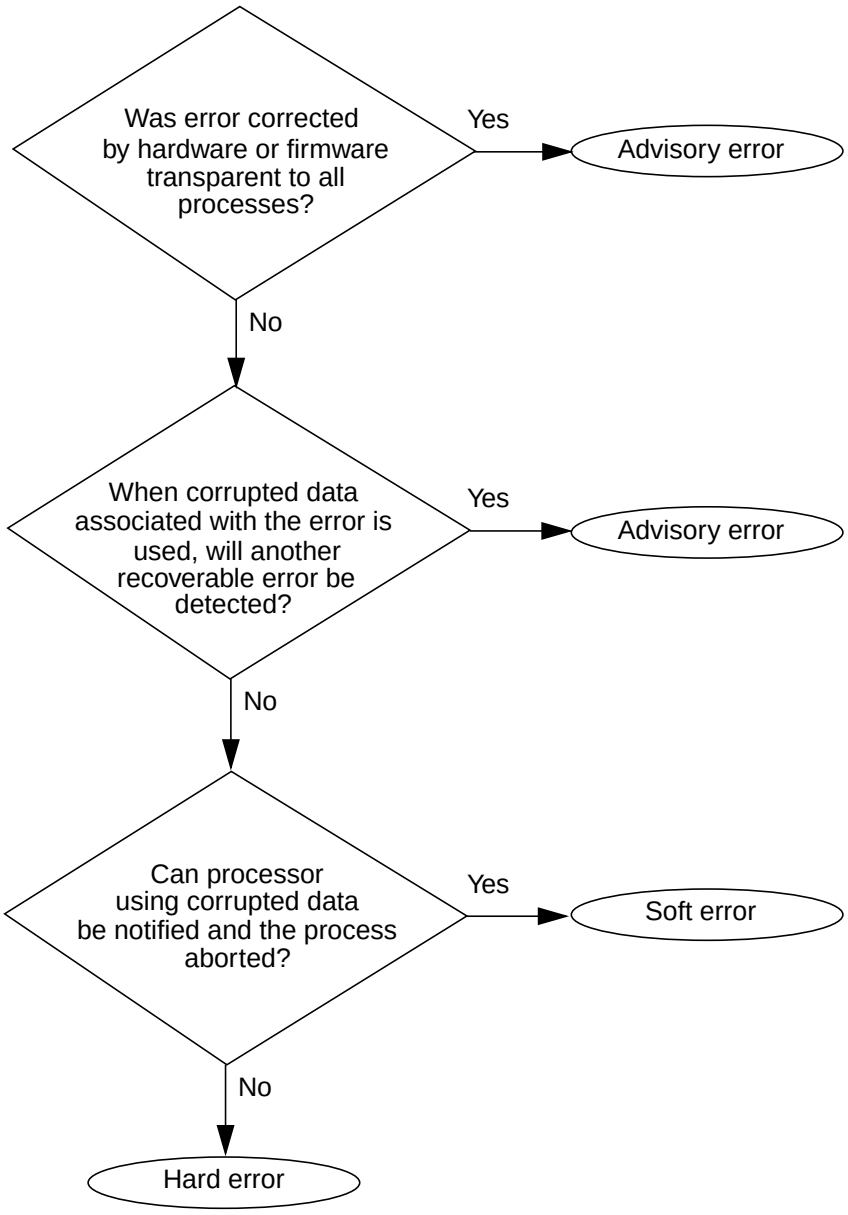
All error CSRs are locked (or frozen) when a hard error is detected so that additional errors caused by the propagation of the hard error are not logged. Usually, only one controller fails in the error condition. If more than one controller detects a hard error, however, the state in the EMUC and clock phase information in each chip indicate the chip that first detected the hard error.

When a hard error occurs, the system must be rebooted. The failed hardware can be deconfigured, as part of reboot, to allow the system to quickly resume operation (possibly with degraded performance) in the presence of broken hardware.

The hard error is logged in the EMUC System Hard Error register. This register logs the first hard error detected, allowing isolation to the controllers or group of controllers that detected the error first.

Figure 81 illustrates the characteristics of the three error types: soft, advisory, and hard.

Figure 81 Determining error types



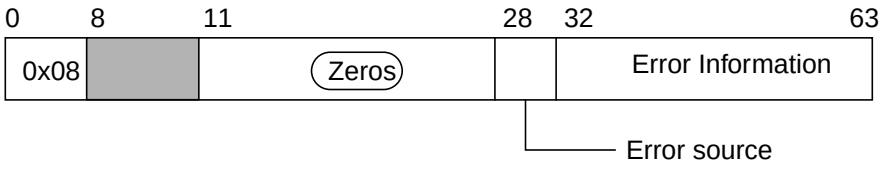
Error responses

When an error occurs during a response-expected data request, the requested data is not returned. Instead, an error response is returned to the requestor. This type of error is called a soft error and is logged as such in the appropriate CSR of the requesting controller.

The response contains information that specifies the detecting controller and the detected error condition or error code; it does not contain data. After receiving an error response, the EPAC logs the error and returns a directed error response packet to the requesting processor or EPIC (in the case of an I/O request). The processor logs the error information in its SADD_LOG register and the EPIC logs the error information in its internal CSRs. Error recovery software can then read these CSRs and take appropriate action.

Figure 82 shows the format for the processor SADD_LOG after an error response.

Figure 82 SADD_LOG after error response



The *Error Source* field (bits 28:31) indicates one of the sources as shown in Table 27.

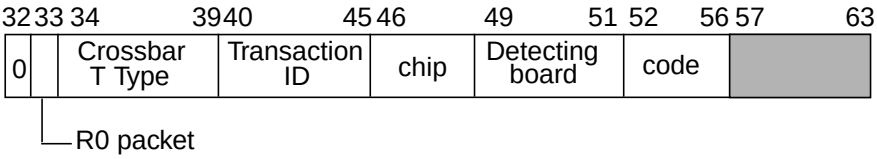
Table 27 SADD_LOG error source field definition

Error source field	Source
0x0	Hyperplane crossbar 0 input
0x1	Hyperplane crossbar 1 input
0x2	Hyperplane crossbar 0 output
0x3	Hyperplane crossbar 1 output
0x4	Runway input
0x5	EPAC CSRs
0x6-0xf	Reserved

The value in the *Error Information* field (bits 32:63) depends on the source of the error response.

If the error source field indicates the response was from either Hyperplane crossbar inputs (that is, external to the EPAC), the Error Information field (bits 32:63) contains the information shown in Figure 83.

Figure 83 EPAC error response information when received from either crossbar input



The subfields and bits of the Error information field are defined as follows:

- *R0* bit (bit 33)—Indicates that the intended response in the crossbar was an R0 packet.
- *Crossbar T type* field (bits 34:39)—Specifies the transaction type of the intended Hyperplane crossbar response.
- *Transaction ID* field (bits 40:45)—Specifies the transaction ID of the intended response.
- *Detecting chip* field (bits 46:48)—Specifies the controller that detected the error.
- *Detecting board* field (bits 49:51)—Specifies the instance of the controller that detected the error (there are eight instances of each controller that can return an error response).
- *Error code* field (bits 52:56)—Specifies the error condition detected by the chip that sent the error response.

If the error source field indicates the response was from the Hyperplane crossbar output logic (internal to the EPAC), the *Error information* field (bits 32:63) contains the transaction ID (TID) of the outgoing transaction. If the *error source* field indicates an error detected by the EPAC Runway bus input logic, the *Error information* field (bits 32:63) contains the type of runway error. If the error source field indicates an error in one of the EPAC CSRs, the *Error information* field (bits 32:63) contains the CSR error code.

Error handling CSRs

Most controllers contain at least one of the following registers:

- Error Cause
- Error Address
- Error Information
- Error Configuration

These registers are accessible through load and store instructions and diagnostic scan.

The Error Cause register logs multiple errors. It has a sticky bit for every possible error condition that can be detected for the controller. Because an error condition can occur under multiple circumstances, a different bit exists in the Error Cause register for each unique circumstance.

The Error Address register contains the address of any error that can be isolated to a specific address. This register is loaded or held under the same error conditions as the Error Information register.

The Error Information register contains error recovery or diagnostic information. This register contains the type of error (listed in increasing severity):

- None (00)
- Advisory (01)
- Soft (10)
- Hard (11)

It also contains the error number for advisory or soft errors. The error number is undefined for hard errors. If an error is detected of the same or lower severity as that already stored in the register, the Error Information register is not overwritten, but the Multiple Error (M) bit is set. If an error type of greater severity is detected, the Error Information register is overwritten with new error information. When error information is overwritten, the Overwritten (O) bit in the register is set.

The Error Configuration register contains two bits for every bit in the Error Cause register. They are encoded as follows:

- 00—Disable the error
- 01—Treat as an advisory error
- 10—Treat as a soft error
- 11—Treat as a hard error

These bits control updating of the Error Information and Error Address registers. They have no effect on the controller behavior except for the following conditions:

- If an error is disabled (bit-pair value is 00) and the controller can still do something meaningful toward completing the operation, the error is ignored. For example, a chip might ignore an address alignment error and assume a certain address when the error is disabled. If no behavior makes sense when an error occurs (that is, the error cannot be ignored), then the disable has no effect on chip operation.
- If the controller scan ring option stop_on_hard bit is set and an error is configured as a hard error (bit-pair value is 11), registers containing information associated with the error must be held. This allows access to the information through scan.

Processor error detection

The processor detects errors that occur during transfers to and from its Runway bus and cache interfaces. It logs these errors and invokes either an LPMC or an HPMC. An LPMC is similar to a trap and allows the process to be restarted. An HPMC is usually fatal to the process, but it may not require a system reboot. Error handling code determines if a processor detected error is treated as advisory, soft or hard.

When errors occur outside the processor that result in an error response to the processor, error information is stored in the processor SADD_LOG register. For timeout errors occurring during noncoherent load or fetch operations, the address associated with the error is stored in the Read Short Logging register. Miscellaneous diagnostic registers contain information about cache parity errors.

EPAC error detection

When the EPAC receives an error response from the Hyperplane crossbar destined for a processor, it sends a directed error, followed (in most cases) by a dummy response to the requesting processor. The error response informs the processor that a soft error was detected during its request and forces the processor to invoke an HPMC. There is no Error Address register in the EPAC. When the EPAC receives an error response destined for the EPIC, it forwards it as an error response packet on the EPIC interface.

When detecting an error, the EPAC stores addresses and other error information in two databases. The information in these databases is accessible with loads and stores. When errors are detected, the EPAC copies information from the database into certain error CSRs.

If an error response to a processor is received or a parity error is detected on the Hyperplane crossbar during a response to a processor, the EPAC copies the error information in the database and into the error CSRs.

If a timeout transaction is detected on the Runway bus, the information corresponding to the timed out response is copied into error CSRs.

If an error is encountered during a message or copy operation, a processor is interrupted, and the detected error condition is logged in the operation status queue.

The EPAC has an interface to Utilities board functions by way of the core logic bus. Processors receive interrupts, fetch instructions, and log error information over the bus. Access to the bus is unaffected by most errors, including a large percentage of hard errors, allowing the processors to perform error logging and recovery.

ERAC error detection

The ERAC routes Hyperplane crossbar packets between the EPAC and the EMAC, checking parity on these packets to and from internal queues. It does not regenerate parity, but passes received parity through queues to its output ports. When the ERAC detects an error, it is logged in the Error Cause and Error Information registers.

There is no Error Address register in the ERAC. The ERAC does not detect when an address is being sent in a packet; therefore, it cannot log the address in an Error Address register.

EMAC error detection

The EMAC contains memory error detection and correction hardware that corrects and logs single-bit errors. When a single-bit error occurs, an interrupt is sent to a designated processor. Single-bit errors on memory data persist in memory and must be scrubbed by error handling software. When the EMAC detects a multibit error on a memory tag, the EMAC generates a hard error. If it detects a multibit error on requested memory data, it sends a parity error with the data to the requestor. If the EMAC detects a multibit error on a read-modify-write operation, it writes bad ECC (that results in a multibit error) to the line to notify any potential users the data line is corrupted.

A CSR map

Table 28 lists the CSRs in the V-Class server.

Table 28 V-Class server CSR map

40-Bit physical address	CSR space	CSR register name
0xF0 0000 0000 - 0xF0 FFFF FFFF	Core Logic	
0xF0 xx00 0000 - 0xF0 xx7F FFFF	PDC EEPROM	V-Class implements 4 MBytes (0x0-0x1FFFFF)
0xF0 xx80 0000 - 0xF0 xxBF FFFF	SRAM	V-Class implements 128 KBytes (0x820000-0x81FFFF)
0xF0 xxC0 0000 - 0xF0 xxC0 FFFF	EPUC byte access	EPUC CSR space
0xF0 xxC0 0000		Interrupt Status register
0xF0 xxC0 0004		Interrupt Enable register
0xF0 xxC0 0008		Interrupt Force register
0xF0 xxC0 000C		EPAC Exist register
0xF0 xxC0 0010		EPUC Revision register
0xF0 xxC0 0014		EPUC Error register
0xF0 xxC1 0000 - 0xF0 xxCF FFFF	EMUC	EMUC CSR space
0xF0 xxC1 0000	Half Word Access	Processor Report register
0xF0 xxC1 0004		Processor Semaphore register
0xF0 xxC1 0008		ERAC Scan Data register
0xF0 xxC1 000C		ERAC Scan Control register
0xF0 xxC1 0010		System Hard Error register
0xF0 xxC1 0014		System Hard Error Enable register

40-Bit physical address	CSR space	CSR register name
0xF0 xxC1 0018		System Hard Error Control register
0xF0 xxC1 001C		Error Cause register
0xF0 xxC1 0020		Environment Error A register
0xF0 xxC1 0024		Environment Error B register
0xF0 xxC1 0028		Environment Error C register
0xF0 xxC1 002C		Environment Control register
0xF0 xxC1 0030		Reset register
0xF0 xxD0 0000 - 0xF0 xxD2 FFFF	M48T35Y	Nonvolatile Ram and Real Time Clock
0xF0 xxD0 0000 - 0xF0 xxD0 7FF7	Byte, Half, Word or Double Word Access	Nonvolatile SRAM
0xF0 xxD0 7FF8		RTC Control register
0xF0 xxD0 7FF9		RTC Seconds register
0xF0 xxD0 7FFA		RTC Minutes register
0xF0 xxD0 7FFB		RTC Hour register
0xF0 xxD0 7FFC		RTC Day register
0xF0 xxD0 7FFD		RTC Date register
0xF0 xxD0 7FFE		RTC Month register
0xF0 xxD0 7FFF		RTC Year register
0xF0 xxD3 0000 - 0xF0 xxD4 5FFF	83932B Sonic Ethernet	Ethernet Interface Chip

40-Bit physical address	CSR space	CSR register name
0xF0 xxD3 0000	Half Word Access	Command register
0xF0 xxD3 0004		Data Configuration register
0xF0 xxD3 0008		Receive Control register
0xF0 xxD3 000C		Transmit Control register
0xF0 xxD3 0010		Interrupt Mask register
0xF0 xxD3 0014		Interrupt Status register
0xF0 xxD3 0018		Upper Transmit Descriptor address
0xF0 xxD3 001C		Current Transmit Descriptor address
0xF0 xxD3 0020		Transmit Packet Size register
0xF0 xxD3 0024		Transmit Fragment Count register
0xF0 xxD3 0028		Transmit Start address 0 register
0xF0 xxD3 002C		Transmit Start address 1 register
0xF0 xxD3 0030		Transmit Fragment Size register
0xF0 xxD3 0034		Upper Receive Descriptor address
0xF0 xxD3 0038		Current Receive Descriptor address
0xF0 xxD3 003C		Current Receive Buffer address 0 register
0xF0 xxD3 0040		Current Receive Buffer address 1 register
0xF0 xxD3 0044		Remaining Buffer Word Count 0 register
0xF0 xxD3 0048		Remaining Buffer Word Count 1 register
0xF0 xxD3 004C		End of Buffer Word Count register
0xF0 xxD3 0050		Upper Receive Resource address register
0xF0 xxD3 0054		Resource Start address
0xF0 xxD3 0058		Resource End address

40-Bit physical address	CSR space	CSR register name
0xF0 xxD3 005C		Resource Read Pointer
0xF0 xxD3 0060		Resource Write Pointer
0xF0 xxD3 0064		Temporary Receive Buffer address 0 register
0xF0 xxD3 0068		Temporary Receive Buffer address 1 register
0xF0 xxD3 006C		Temporary Buffer Word Count 0 register
0xF0 xxD3 0070		Temporary Buffer Word Count 1 register
0xF0 xxD3 007C		Last Link Field address
0xF0 xxD3 0080		Temporary Transmit Descriptor address
0xF0 xxD3 0084		CAM Entry Pointer
0xF0 xxD3 0088		CAM address Port 2 register
0xF0 xxD3 008C		CAM address Port 1 register
0xF0 xxD3 0090		CAM address Port 0 register
0xF0 xxD3 0094		CAM Enable register
0xF0 xxD3 0098		CAM Descriptor Pointer register
0xF0 xxD3 009C		CAM Descriptor Count register
0xF0 xxD3 00A0		Silicon Revision register
0xF0 xxD3 00A4		Watchdog Timer 0 register
0xF0 xxD3 00A8		Watchdog Timer 1 register
0xF0 xxD3 00AC		Receive Sequence Count
0xF0 xxD3 00B0		CRC Error Tally register
0xF0 xxD3 00B4		FAE Tally register
0xF0 xxD3 00B8		Missed Packet Tally register
0xF0 xxD3 00BC		Maximum Deferral Timer register

40-Bit physical address	CSR space	CSR register name
0xF0 xxD3 00FC		Data Configuration register 2
0xF0 xxD4 6000 - 0xF0 xxD4 9FFF	16552 DUART Serial Port 0	Serial Port 0, used for console communication
0xF0 xxD4 6000	Byte Access	Receiver Buffer register/ Transmitter Holding register/ LSB Divisor Latch
0xF0 xxD4 6004		Interrupt Enable register/ MSB Divisor Latch
0xF0 xxD4 6008		Interrupt Identification register/ FIFO Control register
0xF0 xxD4 600C		Line Control register
0xF0 xxD4 6010		Modem Control register
0xF0 xxD4 6014		Line Status register
0xF0 xxD4 6018		Modem Status register
0xF0 xxD4 601C		Scratch Pad register (SCR)
0xF0 xxD4 A000 - 0xF0 xxD4 BFFF	16552 DUART Serial Port 1	Serial Port 0
0xF0 xxD4 A000	Byte Access	Receiver Buffer register/ Transmitter Holding register/ LSB Divisor Latch
0xF0 xxD4 A004		Interrupt Enable register/ MSB Divisor Latch
0xF0 xxD4 A008		Interrupt Identification register/ FIFO Control register
0xF0 xxD4 A00C		Line Control register
0xF0 xxD4 A010		Modem Control register
0xF0 xxD4 A014		Line Status register

40-Bit physical address	CSR space	CSR register name
0xF0 xxD4 A018		Modem Status register
0xF0 xxD4 A01C		Scratch Pad register
0xF0 xxD4 C000 - 0xF0 xxD4 FFFF	16552 DUART Parallel Port	Parallel Port, used for communication
0xF0 xxD4 C000	Byte Access	Read Data/ Write Data
0xF0 xxD4 C004		Read Status
0xF0 xxD4 C008		Read Control/ Write Control
0xF4 0000 0000 - 0xF7 FFFF FFFF	Nonaccelerated I/O	
0xF8 0000 0000 - 0xFB FFFF FFFF	Accelerated I/O	
0xFC 0000 0000 - 0xFC 0000 FFFF	EPAC 0	EPAC CSR space
0xFC 0000 0000	EPAC 0, Page 0	System Configuration register
0xFC 0000 0008		EPAC Chip Configuration register
0xFC 0000 0010		EPAC Core Logic Interrupt Delivery register 0
0xFC 0000 0018		EPAC Core Logic Interrupt Delivery register 1
0xFC 0000 0020		Memory Board Configuration register
0xFC 0000 0080		EPAC Error Cause register 0
0xFC 0000 0088		EPAC Error Info register
0xFC 0000 0098		EPAC Error Configuration register 0
0xFC 0000 00A0		EPAC Error Configuration register 1
0xFC 0000 00A8		EPAC Error Cause register 1
0xFC 0000 0300		Time-of-Century Configuration register

40-Bit physical address	CSR space	CSR register name
0xFC 0000 1308	EPAC 0, Page 1	Time-of-Century Count register
0xFC 0000 2000	EPAC 0, Page 2 Processor 0 Specific	Processor 0 Processor Configuration register
0xFC 0000 2010		Processor 0 CSR Operation Context register
0xFC 0000 2018		Processor 0 CSR Operation address register
0xFC 0000 2020		Processor 0 Fetch and Increment address
0xFC 0000 2028		Processor 0 Fetch and Decrement address
0xFC 0000 2030		Processor 0 Fetch and Clear address
0xFC 0000 2038		Processor 0 Noncoherent Read address
0xFC 0000 2040		Processor 0 Noncoherent Write address
0xFC 0000 2060		Processor 0 Coherent Increment address
0xFC 0000 2100		Processor 0 DM Input Command register
0xFC 0000 2110		Processor 0 DM Source Physical Page Frame register
0xFC 0000 2118		Processor 0 DM Source Offset register
0xFC 0000 2120		Processor 0 DM Destination Physical Page Frame register
0xFC 0000 2128		Processor 0 DM Destination Offset register
0xFC 0000 2130		Processor 0 DM Operation Status Queue register
0xFC 0000 2200		Processor 0 Performance Monitor Memory Access Count 0 register
0xFC 0000 2208		Processor 0 Performance Monitor Memory Access Count 1 register
0xFC 0000 2210		Processor 0 Performance Monitor Memory Access Latency register

40-Bit physical address	CSR space	CSR register name
0xFC 0000 3000	EPAC 0, Page 3 Processor 1 Specific	Processor 1 Processor Configuration register
0xFC 0000 3010		Processor 1 CSR Operation Context register
0xFC 0000 3018		Processor 1 CSR Operation address register
0xFC 0000 3020		Processor 1 Fetch and Increment address
0xFC 0000 3028		Processor 1 Fetch and Decrement address
0xFC 0000 3030		Processor 1 Fetch and Clear address
0xFC 0000 3038		Processor 1 Noncoherent Read address
0xFC 0000 3040		Processor 1 Noncoherent Write address
0xFC 0000 3060		Processor 1 Coherent Increment address
0xFC 0000 3100		Processor 1 DM Input Command register
0xFC 0000 3110		Processor 1 DM Source Physical Page Frame register
0xFC 0000 3118		Processor 1 DM Source Offset register
0xFC 0000 3120		Processor 1 DM Destination Physical Page Frame register
0xFC 0000 3128		Processor 1 DM Destination Offset register
0xFC 0000 3130		Processor 1 DM Operation Status Queue register
0xFC 0000 3200		Processor 1 Performance Monitor Memory Access Count 0 register
0xFC 0000 3208		Processor 1 Performance Monitor Memory Access Count 1 register
0xFC 0000 3210		Processor 1 Performance Monitor Memory Access Latency register

40-Bit physical address	CSR space	CSR register name
0xFC 0001 0000 - 0xFC 0001 FFFF	EPIC	EPIC CSR space
0xFC 0001 0000	EPIC 0, Page 0	System Configuration register (Reserved on EPIC)
0xFC 0001 0008		EPIC Chip Configuration register
0xFC 0001 0010		PCI Master Configuration register
0xFC 0001 0018		PCI Master Status register
0xFC 0001 0020		EPIC Channel Builder register
0xFC 0001 0080		EPIC Error Cause register
0xFC 0001 0088		EPIC Error Info register
0xFC 0001 0090		EPIC Error Address register
0xFC 0001 0098		EPIC Error Configuration register
0xFC 0001 00A0		EPIC Interrupt Configuration register
0xFC 0001 00A8		EPIC Interrupt Source register
0xFC 0001 00B0		EPIC Interrupt Enable register
0xFC 0001 0100		PCI Slot 0 Configuration register
0xFC 0001 0108		PCI Slot 0 Status register
0xFC 0001 0110		PCI Slot 0 Interrupt Configuration register
0xFC 0001 0118		PCI Slot 0 Synchronization register
0xFC 0001 0120		PCI Slot 1 Configuration register
0xFC 0001 0128		PCI Slot 1 Status register
0xFC 0001 0130		PCI Slot 1 Interrupt Configuration register
0xFC 0001 0138		PCI Slot 1 Synchronization register
0xFC 0001 0140		PCI Slot 2 Configuration register

40-Bit physical address	CSR space	CSR register name
0xFC 0001 0148		PCI Slot 2 Status register
0xFC 0001 0150		PCI Slot 2 Interrupt Configuration register
0xFC 0001 0158		PCI Slot 2 Synchronization register
0xFC 0001 0160		PCI Slot 3 Configuration register
0xFC 0001 0168		PCI Slot 3 Status register
0xFC 0001 0170		PCI Slot 3 Interrupt Configuration register
0xFC 0001 0178		PCI Slot 3 Synchronization register
0xFC 0002 0000		Processor 0 External Interrupt Request register
0xFC 0003 0000		Processor 1 External Interrupt Request register
0xFC 0004 0000 - 0xFC 0004 FFFF	EMAC	EMAC CSR space
0xFC 0004 0000	EMAC 0, Page 0	System Configuration register
0xFC 0004 0008		EMAC Chip Configuration register
0xFC 0004 0020		Memory Row Configuration register
0xFC 0004 0028		Unprotected Memory Region register
0xFC 0004 0080		EMAC Error Cause register
0xFC 0004 0088		EMAC Error Info register
0xFC 0004 0090		EMAC Error address register
0xFC 0004 0098		EMAC Error Configuration register 0
0xFC 0004 00A0		EMAC Error Configuration register 1
0xFC 0004 00B0		EMAC Error Interrupt register
0xFC 0004 0100		Message Reception Area Configuration register
0xFC 0004 0110		Message Reception Area Available Offset register
0xFC 0004 0118		Message Reception Area Occupied Offset register

40-Bit physical address	CSR space	CSR register name
0xFC 0004 0120		Message Completion Queue Configuration register
0xFC 0004 0128		Message Completion Queue Reserve Offset register
0xFC 0004 0130		Message Completion Queue Write Offset register
0xFC 0004 0138		Message Completion Queue Read Offset register
0xFC 0004 0140		Message Completion Dequeue address
0xFC 0004 0200		Diagnostic Address register
0xFC 0004 0208		Diagnostic Data register
0xFC 0004 0220		Diagnostic Read Memory Data address
0xFC 0004 0228		Diagnostic Write Memory Data address
0xFC 0004 0230		Diagnostic Read Memory ECC address
0xFC 0004 0238		Diagnostic Write Memory ECC address
0xFC 0004 0240		Diagnostic Initialize Memory address
0xFC 0004 0248		Diagnostic Scrub Memory address
0xFC 0004 F148	EMAC 0, Page F	Message Completion Enqueue address
0xFC 0004 F150		Message Allocation address
0xFC 0006 0000 - 0xFC 0006 FFFF	Reserved	
0xFC 0007 0000 - 0xFC 0007 FFFF	Reserved	
0xFC 0008 0000 - 0xFC 000F FFFF	Hyperplane crossbar Port 1	
0xFC 0010 0000 - 0xFC 0017 FFFF	Hyperplane crossbar Port 2	

40-Bit physical address	CSR space	CSR register name
0xFC 0018 0000 - 0xFC 001F FFFF	Hyperplane crossbar Port 3	
0xFC 0020 0000 - 0xFC 0027 FFFF	Hyperplane crossbar Port 4	
0xFC 0028 0000 - 0xFC 002F FFFF	Hyperplane crossbar Port 5	
0xFC 0030 0000 - 0xFC 0037 FFFF	Hyperplane crossbar Port 6	
0xFC 0038 0000 - 0xFC 003F FFFF	Hyperplane crossbar Port 7	

Glossary

absolute address An address that does not undergo virtual-to-physical address translation when used to reference memory or the I/O register area.

address A number used by the operating system to identify a storage location.

address space Memory space, either physical or virtual, available to a process.

advisory error Errors that are usually corrected by hardware or firmware

architecture The physical structure of a computer's internal operations, including its registers, memory, instruction set, and I/O structure.

barrier synchronization A control mechanism used in parallel programming that ensures all processors have completed the prior operation before continuing with the next operation.

block A group of data containing a fixed number of bytes.

block TLB A type of TLB entry that translates many contiguous virtual pages to an equal number of contiguous physical pages.

boot The procedure by which a program is initiated the first time. Typically, a bootstrap is performed when power is first applied to the processor.

buffer A temporary storage area. Several types of buffers are used in computer systems, in both hardware and software. The most common types of buffers are those maintained by a computer operating system to mediate between processes and I/O devices.

bus A data path shared by several components within a computer system.

cache See cache memory.

cache memory A small, high-speed buffer memory used in modern computer systems to hold temporarily those portions of the contents of the main memory that are, or believed to be, currently in use.

check A type of interruption caused by the detection of an internal hardware detected malfunction.

coherency A term frequently applied to caches. If a data item is referenced by a particular processor on a multi-processor

system, the data is copied into that processor cache and is updated there if the processor modifies the data. The state that is achieved when both processors' caches always have the latest value for the data is called cache coherency.

crossbar See Hyperplane crossbar

CSR Control and status register. A software-addressable hardware register used to hold control or state information.

data mover Hardware that routes messages and copies data between memory.

direct memory access (DMA) A procedure or method defined for gaining direct access to main storage and achieving data transfers without involving the processor.

DMA See direct memory access.

EEPROM See electrically erasable programmable read-only memory.

electrically erasable programmable read-only memory (EEPROM) A read-only memory module that can be programmed repeatedly by first erasing the previous contents of the memory module. Unlike the EPROM, the EEPROM can be reprogrammed without removal from the circuit board by applying an erase signal to the device.

error code The status returned by a function call.

ECUB Exemplar Core Utilities board. Located on the ENRB, the ECUB provides system booting and testing functions.It connects to the core logic bus, the system environmental sensors, and other test points, as well as the optional teststation.

EMAC Exemplar Memory Access Controller. The gate array that controls system memory. It interfaces to the routing array controller (ERAC) Each EMAC controls four banks of memory, allowing up to 32 banks in an eight-EMAC system.

EPAC Exemplar Processor agent controller. The gate array that interfaces to pairs of PA-RISC processors.

EPIC Exemplar PCI Interface Controller. The heart of the S-Class and X-Class I/O subsystem. The EPIC connects to the PCI bus and provides and interface between I/O devices and the EPAC.

ERAC Exemplar Routing Array Controller. The ASIC (four required) used to in the system Hyperplane crossbar. The crossbar provides an interface between the processors and I/O devices and system memory.

error correction code (ECC) Code used to decide which bit of a memory read operation is in error.

exception A hardware-detected event that disrupts the running of a program, process, or system. See also fault.

fault A type of interrupt caused by an instruction that requests a legitimate action that cannot be carried out immediately due to a system problem.

shared memory A memory architecture in memory can be accessed by all processors in the system. This architecture can also support virtual memory. This type of memory is sometimes referred to as shared virtual memory or global virtual memory.

hard error An uncorrectable data error.

Hyperplane crossbar A switching device used in multiprocessor, shared-memory computer systems that connects processors to the various banks of memory in the system.

HPMC High priority machine check. Indicates that a process error occurred and that the process cannot continue. The system requires rebooting after an HPMC.

instruction cache (Icache) Memory used to hold frequently accessed instructions.

interface A physical path between any two modules or systems.

interleaved memory Memory that is divided into multiple banks to permit concurrent memory accesses. The number of separate memory banks is referred to as the memory interleave. Programs can optimize memory accesses by using stride intervals so that each of the banks can be refreshed between memory accesses.

interrupt An occurrence that changes the normal flow of instruction execution. An interrupt originates from hardware, such as an I/O device. See also maskable interrupt.

interval timer An interval timer is used to generate an interrupt based on the passage of time.

JTAG Joint Test Action Group. Formerly a group of European and later American companies that developed a boundary scan technique to facilitate in-circuit testing and functionality testing of circuit boards. It was handed off to the IEEE and refined as IEEE Standard 1149.1.

latency The time delay between the issuing of an instruction and the completion of the operation. A common metric used for comparing parallel processor systems is the latency of coherent memory.

LPMC Low priority machine check.It is similar to a processor trap in that it is not fatal to a process.

main memory See physical memory.

maskable interrupt An interrupt to which the operating system may choose not to respond.

message passing A type of programming in which program modules (often running on different processors or different hosts) communicate with each other by means of system library calls that package, transmit, and receive data.

move-in The operation of bringing information from memory into a cache.

node A complete system that consists of a set of up to 16 processors and up to 64 Gbytes of physical memory organized as a symmetric multiprocessor (SMP) running a single image of the operating system microkernel.

noncoherent memory reference A memory reference that 1) does not cause a cache move-in, or 2) causes a cache move-in, but fails to obey cache coherency rules.

PA-RISC The Hewlett-Packard precision architecture reduced instruction set computer. A RISC instruction set is easy to decode in hardware and for which a compiler can generate highly optimized code.

packet A group of related items. A packet may refer to the arguments of a subroutine or to a group of bytes that is transmitted over a network.

page The unit of logical memory controlled by the memory management algorithms. A page in the V-Class server is 4 Kbytes (4,096) contiguous bytes.

page frame Unit of physical (main) memory in which pages are placed. Referenced and modified bits associated with each page frame aid in memory management.

PDIR Physical page directory. PA-RISC processors that implement hardware TLB miss handlers may fetch TLB entries from a PDIR entry in the event of a TLB miss. The PDIR serves as a cache of virtual-to-physical page translations and is maintained by the operating system.

physical address A unique identifier that selects a particular device from the set of all devices connected to a particular bus.

physical address space The set of possible addresses for a particular bus.

physical memory Memory devices, usually RAM, connected as a subsystem that provide fast-access storage for the operating system, applications, and data.

process The fundamental unit of a program that is managed by the job scheduler. A collection of one or more execution streams within a single logical address space; an executable program. A process is made up of one or more threads.

register A hardware entity that contains an address, operand, or instruction status information.

reset The process of establishing a known state in a machine register.

runway bus The data interface of the Hewlett-Packard PA-8200 processor. It allows multiprocessor systems to interface to memory and I/O without additional components.

SDRAM Synchronous Dynamic Random Access Memory.

semaphore A group of bits associated with data structures that act as a flag to all processors to synchronize the threads of a multiple-thread process. See also synchronization.

server A process that fulfills a request issued by a client process, and transmits a response back to the client.

SIMM Single Inline Memory Module.

snooping Externally flushing or invalidating a cache line from a processor cache. The flushing or invalidating transaction is issued by the processor agent and is referred to as a “snoopy transaction.” This action occurs when one processor in the system loads or stores to a dirty line in the cache of another processor, or when a processor stores to a line that is shared by one or more processors in the system.

soft error Correctable single-bit memory error. May further be defined as transient (non-reproducible) or stuck (reproducible).

space A contiguous range of virtual addresses within the system wide virtual address space.

synchronization A way to keep two threads from accessing the same critical region simultaneously. You can synchronize programs using compiler directives, thread library calls, or assembly-language instructions. You do so, however, at the cost of additional overhead; synchronization may cause at least one processor to wait for another.

system console The terminal or workstation that serves as a communication device between the system manager and the computer system. On the V-Class server, the teststation serves as the system console.

TLB Translation Lookaside Buffer. A hardware structure in each processor of the V-Class server that contains the information necessary to translate

a virtual memory reference to a physical page and to validate memory accesses.

TOC Time of Century. Used register to time stamp trace data and interprocess messages.

teststation The workstation that is used to diagnose problems and install system software.

thread An independent execution stream that is fetched and executed by a processor. One or more threads, each of which can execute on a different processor, make up each process. Threads are created and terminated by instructions that can be automatically generated by compilers, inserted by adding compiler directives to source code, or coded explicitly in Fortran, C, or C++ programs.

thread-private or thread-specific Data that is accessible by a single thread only (not shared among the threads constituting a process). Thread-specific data allows the same virtual address to refer to different physical memory locations.

trap A type of interrupt caused when either the function requested by the current instruction cannot or should not be carried out, or system intervention is desired by the user before or after the current instruction is executed. Typically, this condition is a result of unexpected arithmetic results.

virtual address An address used by a program to access data or instructions. The V-Class server maps each virtual address to physical memory location.

virtual alias Two different virtual addresses that map to the same physical memory address.

wall-clock time The time an application requires to complete its processing. If an application starts running at 1:00 p.m. and finishes at 5:00 a.m. the following morning, its wall-clock time is sixteen hours.

wired-down The term that applies to virtual-to-physical address translation indicating that the two addresses remain the same after translation.

Index